68' MICRO JOURNAL

Australia	A $ 4.75	New Zealand	NZ $ 6.50	**$2.95 USA**
Singapore	S $ 9.45	Hong Kong	H $23.50	
Malaysia	M $ 9.45	Sweden	30:-SEK	

Motorola S-50 BUS•VME•MACINTOSH & Other 68XXX Systems

6809 68008 68000 68010 68020 68030

OS-9 The Magazine for Motorola CPU Devices *For Over a Decade!* *A User Contributor Journal* FLEX SK•DOS

This Issue:

And Lots More!

VOLUME VIII ISSUE XII • Devoted to the 68XX User • December 1986

"Small Computers Doing Big Things"

PHOTO COURTESY: NASA

WE HAVE MOTOROLA COVERED

GESSBS-4 $316*

1 MHz 6809 CPU
Sockets for up to 32 Kbytes EPROM
Sockets for up to 16 Kbytes CMOS RAM
One RS 232 serial port
40 TTL Bidirectional I/O lines
4 x 16-bit timers

GESMPU-14 $636*

8 MHz 68010 CPU
Optional 32081 arithmetic unit
Sockets for up to 128 Kbytes EPROM
One RS 232 serial port
4 x 8-bit timers
Real-time clock/calendar and battery

GESMPU-4A $316*

8 MHz 68000 CPU
Sockets for up to 128 Kbytes EPROM
Sockets for up to 64 Kbytes CMOS RAM
One RS 232 serial port
Three 16-bit timers

SOFTWARE

OS-9® , PDOS® , CP/M 68K® , Editor-Assembler, Basic-Pascal-C compilers, FORTH.

Gespac

USA - CANADA
100 West Hoover Ave.
Mesa, AZ 85202
Tel. (602) 962-5559
Telex 386575

INTERNATIONAL
3,chemin des Aulx
CH-1228 Geneva
Tel. (022) 713400
Telex 429989

* 100 piece quantities

Ⓜ is a registered trademark of Motorola Inc.

GMX Micro-20 prices

MICRO 20 (12.5 MHz).....$2565.00
MICRO 20 (16.67 MHz).....$2895.00
8 PORT RS232 BOARD SET (SBC-8S).....$ 498.00
PROTOTYPING BOARD (SBC-MW).....$ 75.00
BACK PANEL PLATE (BPP-PC).....$ 44.00
I/O BUS ADAPTER (SBC-8A).....$ 195.00

QUANTITY DISCOUNTS ARE AVAILABLE ON THE
ABOVE ITEMS AS FOLLOWS:4-9,LESS 5%;
10-24,LESS 10%; 25-99,LESS 20%; 100 UP,LESS 30%.

MC6888IRC12.....$ 295.00
MC6888IRC16.....$ 395.00
SBC ACCESSORY PACKAGE (M20-AP).....$1690.00
For other configurations and options, contact GMX.
MOTOROLA 68020 USERS MANUAL.....$ 18.00
MOTOROLA 68881 USERS MANUAL.....$ 18.00

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GMX
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

GMX S-50 BUS prices

68020 SYSTEM

For the user who appreciates the need for a bus structured system using STATIC RAM and powered by a ferro resonant constant voltage transformer, DMA transfers, high speed MMU, we have the UNIFLEX-VM 68020 development system.
The system CPU provides protection to the system and other users from crashes caused by defective user programs.
The system's intelligent serial I/O processor boards significantly reduce system overhead by handling routine I/O functions.
The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. It allows up to 4 Megabytes of Virtual Memory per user. All systems include 1MB of static RAM, one 3-port intelligent Serial I/O board, DMA Controllers, a 5" 80 track floppy drive.

PRICES

#020 UniFLEX VM with 25MB HD.....$10,980.20
#020 UniFLEX VM with 85MB HD.....$12,480.20

YOU CAN EXPAND THESE 020 SYSTEMS WITH:
60MB STREAMER.....$ 2,400.00
REMOVABLE PACK DRIVE.....$ 1,200.00

INTELLIGENT I/OS
#14 3 Port Serial-30 Pin.....$ 498.14
#13 4 Port Serial-50 Pin.....$ 618.13
#12 Parallel-50 Pin.....$ 538.12

CABLE SETS FOR I/OS
95 Cable Sets Specify Card.....$ 24.95
51 Cent. B.P. Cable for #12 & #44....$ 34.51
53 Cent. Cable Set.....$ 36.53

6809 SYSTEM

The number 39 systems include:#05 CPUwDAT; #19 Classy Chassis; 256K Static RAM; a # 43 2 port serial card & cables; #68 DMA Controller; all necessary cables, power regulators, and filler plates;

System # 39 OS-9 GMX II Dual 80 DSDD...$ 2,998.39
" w19MB.....$ 4,698.39
" w72MB.....$ 6,298.39

The Software included in this System: GMXBUG monitor; FLEX; and OS-9 GMXII. You can software select either FLEX or OS-9. Also includes OS-9 Editor, Assembler, Debugger, BASIC-09, RUNB, RMS, DD, and GMX-VDISK for FLEX.

System # 39 UniFLEX w25MB.....$ 4,698.39
" w85MB.....$ 6,298.39

The UniFLEX Operating System is included.

6809 SYSTEMS USING THE GIMIX III CPU & INTELLIGENT I/O PROCESSOR BOARDS

These System include: GMX6809 CPU III; one #11 3 port Intelligent serial I/O & Cables; #19 Classy Chassis; 256K Static RAM; #60 DMA controller; all necessary cables, power regulators, and filler plates.

System # 79 OS9 GMX III Dual 80 DSDD...$ 4,498.79
" w25MB.....$ 6,498.79
" w85MB.....$ 7,998.79

The # 79 System Software includes: OS9 GMXIII; OS9 Editor, Assembler, Debugger, BASIC 09, RUNB, RMS, DD, RAMdisk, O-FLEX; GMXBUG; FLEX. The GMX Support ROM and the hardware CRC board are exclusive features included in this system.

System # 89 UniFLEX III w25MB.....$ 6,798.39
w85MB.....$ 8,298.39

The UniFLEX GMX III Operating System is included.

A Member of the CPI Family

68 Micro Journal

6800 6809 68000 68010 68020

10 Years of Dedication to Motorola Users

Editorial Staff

Contents

"Contribute *Nothing* - Expect *Nothing*" DMW 1986

COMPUTER PUBLISHING, INC.
"Over a Decade of Service"
"World Wide"

68 MICRO JOURNAL
CPI
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 - Telex 510 600-6630

68 Micro Journal is published 12 times a year by Computer Publishing, Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year: $24.50 USA, Canada & Mexico $34.00 ***a year.***
Others ***add*** **$12.00** ***a year surface, airmail add*** **$48.00** ***a year, USA funds!*** **2 Years $42.50, 3 Years $64.50** ***plus additional postage, for each*** **additional year.**

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number and date, as well as a statement that the material is original and the property of the submitting author. Articles submitted should be on diskette, Macintosh, OS-9 or FLEX format. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please.

Please do not *format with spaces any text indents, chart items, etc. (source listings o.k.) WE will edit in ALL formatting.* Text should be flush left column and use ONLY a carriage return to separate paragraphs or other article text items! MacWrite, FLEX TSC, Stylo formatting acceptable.

Letters & Advertising Copy

Letters to the Editor should be original copy, signed! Letters of gripe as well as praise are acceptable. *We reserve the right to reject any letter to the editor or advertising copy material, for any reason we deem advisable.*

Advertising Rates: Commercial please contact 68 Micro Journal advertising department. Classified ads must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word after the first 15. All classifieds must be pre-paid. No classifieds accepted by telephone.

The VME BUS and OS-9:

Ultimate Software for the Ultimate Bus.

Modularity. Flexibility. High Performance. Future growth. These are probably the prime reasons you chose the VME bus. Why not use the same criteria when selecting your system software? That's why you should take a look at Microware's OS-9/68000 Operating System—it's the perfect match for the VME bus.

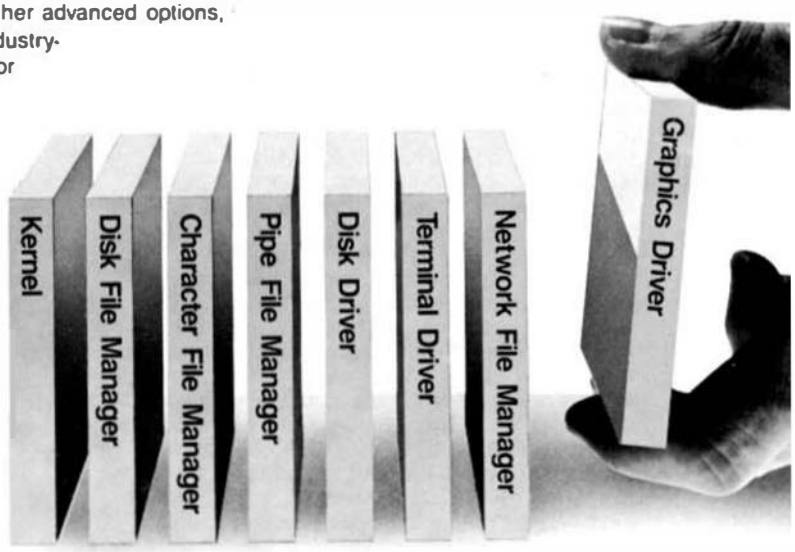
When you're working with VME you must have access to every part of the system. Unlike other operating systems that literally scream KEEP OUT!, OS-9's open architecture invites you to create, adapt, customize and expand. Thanks to its unique modular design, OS-9 naturally fits virtually any system, from simple ROM-based controllers up to large multiuser systems.

And that's just the beginning of the story. OS-9 gives you a complete UNIX-application compatible environment. It is multitasking, real time, and extremely fast. And if you're still not impressed, consider that a complete OS-9 executive and I/O driver package typically fits in less than 24K of RAM or ROM.

Software tools abound for OS-9, including outstanding Microware C, Basic, Fortran, and Pascal compilers. In addition, cross C compilers and cross assemblers are available for VAX systems under Unix or VMS. You can also plug in other advanced options, such as the GSS-DRIVERS™ Virtual Device Interface for industry-standard graphics support, or the OS-9 Network File Manager for high level, hardware-independent networking.

Designed for the most demanding OEM requirements, OS-9's performance and reliability has been proven in an incredible variety of applications. There's nothing like a track record as proof: to date, over 200 OEMs have shipped more than 100,000 OS-9-based systems.

Ask your VME system supplier about OS-9. Or you can install and evaluate OS-9 on your own custom system with a reasonably priced Microware PortPak™. Contact Microware today. We'll send you complete information about OS-9 and a list of quality manufacturers who offer off-the-shelf VME/OS-9 packages.



Modular Hardware Deserves Modular Software

MICROWARE.

Microware Systems Corporation
1866 N.W. 114th Street • Des Moines, Iowa 50322
Phone 515-224-1929 • Telex 910-520-2535

Microware Japan, Ltd.
41-19 Honcho 4-Chome, Funabashi City • Chiba 273,
Japan • Phone 0473 (28) 4493 • Telex 781-299-3122

Micromaster Scandinavian AB
S:t Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
Phone: 018-138595
Telex: 76129

Dr. Rudolf Keil, GmbH
Porphystrasse 15
D-6905 Schriesheim
West Germany
Phone: (0 62 03) 67 41
Telex: 465025

Elsoft AG
Zelweg 12
CH-5405 Baden-Dattwil
Switzerland
Phone: (056) 83-3377
Telex: 828275

Vivaway Ltd.
36-38 John Street
Luton, Bedfordshire, LU1 2JE
United Kingdom
Phone: (0582) 423425
Telex: 825115

Microprocessor Consultants
92 Bynya Road
Palm Beach 2108
NSW Australia
Phone: 02-919-4917

Microdata Soft
97 bis, rue de Colombes
92400 Courbevoie
France
Phone: 1-788-80-80
Telex: 815405

OS-9 is a trademark of Microware and Motorola. PortPak is a trademark of Microware. GSS-Drivers is a trademark of Graphic Software Systems, Inc. VAX and VMS are trademarks of DEC. Unix is a trademark of AT&T.

MUSTANG -020 Super SBC™

DATA-COMP proudly presents the first Under $5000 "SUPER MICRO".

The MUSTANG-020™

MUSTANG-020™

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $5000, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 20 serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The system SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $3000. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based bebugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

DATA-COMP
Installed Systems World-Wide
OVER 10 YEARS OF DEDICATED QUALITY!
CPI A Division of Computer Publishing, Inc.
5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

MUSTANG-020, MUSTANG-08 Benchmarks

All timings by independent consultant	Time - seconds 32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	no register
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	5.1	3.2
68000 OS-9 68K 10 Mhz	6.5	4.0
68008 OS-9 68K 8 Mhz	18.0	9.0
MUSTANG-08 6800E OS-9 68K 10 Mhz	9.8	6.3
MUSTANG-020 68020 OS-9 68K 16 Mhz	2.2	0.88
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	1.8	1.22

```
Main()
    {
        register long i;
        for (i=0; i < 999999; ++i);
    }
```

Estimated MIPS - MUSTANG-020 --- 4.5 MIPS.
Burst to 5 - 10 MIPS: Motorola Specs.

MUSTANG-020™ Software

OS-9

OS-9	$350.00
Basic09	300.00
C Compiler	400.00
Fortran 77	400.00
Microware Pascal	400.00
Consumer Pascal	900.00
Style-Graph	695.00
Style-Spell	195.00
Style-Merge	175.00
Style-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX	$450.00
Screen Editor	150.00
Son-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor+ (see below)	995.00

Option & Expansion

8 Port expansion RS-232 $98.00
(total of 20 serial ports supported)

Expansion for Motorola I/O Channel Modules $195.00

** All Expansion boards:
All expansion boards for old style cabinet will require the 101 expansion cable. Systems ordered with newer PC type cabinet do not require this cable.

101 Expansion Cable $39.95

Sculptor+: We are USA distributors for Sculptor+. Call or write for site or multiple system license & discounts. OEM/Dealer.

Special for complete MUSTANG-020 system buyers - Sculptor+ $695.00. Save $300.00!

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

MUSTANG-020-

FEATURES

- 12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path processor
- 32-bit wide data and address buses, non-multiplexed
- on chip instruction cache
- object code compatible with all 68XXX family processors
- enhanced instruction set - math co-processor interface
- 68881 math hi-speed floating point co-processor (optional)
  direct extension of full 68020 instruction set
  full support IEEE P754, draft 10.0
  transcendental and other scientific math functions
- 2 Megabyte of SIP RAM (512 x 32 bit organization)
- up to 256K bytes of EPROM (64 x 32 bits)
- 4 Asynchronous serial I/O ports standard
  optional to 20 serial ports
  standard RS-232 interface
  optional network interface
- buffered 8 bit parallel port (1/2 MC68230)
  Centronics type pinout
- expansion connector for additional I/O devices
  16 bit data path
  256 byte address space
  2 interrupt inputs
  clock and control signals
  Motorola I/O Channel Modules
- time of day clock/calendar w/battery backup
- controller for 2, 5 1/4" floppy disk drives
  single or double side, single or double density
  35 to 80 track selectable (48-96 TPI)
- SASI interface
- programmable periodic interrupt generator
  interrupt rate from micro-seconds to seconds
  highly accurate time base (5 PPM)
- 5 bit sense switch, readable by the CPU
- hardware single-step capability
- mounts directly to a standard 5 1/4" disk drive

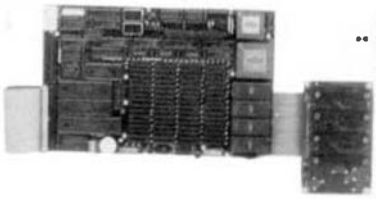
CPI
Mustang-020-
A DATA-COMP
Hi-Speed
Product

Size 8 15/16 x 5 7/8

These *hi-speed* 68020 systems are presently working at NASA, Atomic Energy Commission, other Government Agencies as well as Universities, Business, Labs, and critical applications centers, Worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a *must*!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information.

MUSTANG-020 System component prices - Effective July 1, 1986
Prices subject to change - call for latest quotes.



	MUSTANG-020 (12.50 Mhz)	$2750.00
**	Cabinet (PC or as shown)	$299.95
	5"-80 track floppy DS/DD	$269.95
	Floppy cable	$39.95
	OS-9 68K	$350.00
	Winchester cable	$39.95
	Winchester Drive 25 Mbyte	$895.00
	Xebec H/D controller	$395.00
	Shipping USA UPS	$20.00
	Total:	$5059.80

** **DISCOUNT LIMITED TIME:** **Complete System $1061.00**

Complete System $3998.80

OPTIONS ADD:

UniFLEX	$90.00
MC68881 f/p math processor	$275.00
16.67 Mhz MC68020	$375.00
16.67 Mhz MC68881	$375.00

WE WILL NOT BE UNDERSOLD!

This price subject to increase
Additional MUSTANG systems soon

Note: Current OS-9 (Ver. 1.2) does not address the MC68881 - Future revisions will. If the 68881 is anticipated in the future, it must be ordered with the system, when originally ordered. UniFLEX does support both the enhanced code of the 68020 and 68881 now.
OPTION BOARDS: ** *Option boards to be installed in Mustang-020 cabinets must be ordered with the extension cable.* The cabinet is too tight for direct plugon. Or specify our new PC type cabinet, with initial order.

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9
68XXX
Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR with all the best of PIE. For those who swore by and loved PIE, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

SOFTWARE USER NOTES

A Tutorial Series

By: Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

PIC

Position Independent Code

Last time we discussed the writing of position independent code in Assembler. I presented a couple of programs that were written in PIC, which used the technique of placing variables in, before, or after the code in the program and then using program counter relative addressing to access them. Another approach is to put the variables on one of the processor stacks (or point one of the stack pointers or index registers at them).

I have seen the system stack pointer S use frequently in subroutines for "local" variables, i.e. storage for variables used completely within a subroutine and having no use outside of the subroutine. The technique is to back the stack pointer up by enough locations to provide for the local variables, and then before the RTS, to restore the stack pointer to be pointing at the return address. Listing 1, included here is a subroutine to convert a hexadecimal byte into two ascii characters that represent its value. That is, for example to convert the binary value 00101110 to the two ascii characters 2E, which represent its value.

```
*
* SUBROUTINE TO CONVERT A BYTE INTO TWO CHARACTERS THAT
* REPRESENT ITS HEXADECIMAL VALUE
*
* BYTE PASSED TO SUBROUTINE IN ACCA, TWO CHARACTERS RETURNED
* IN ACCA AND ACCB, HIGH ORDER IN ACCA.   E.G. $3A PASSED IN
* ACCA.  EXITS WITH ACCA AND ACCB CONTAINING $33 AND $41
* RESPECTIVELY, THE ASCII CODES FOR "3" AND "A".
*
MAKHEX LEAS -2,S MAKE ROOM FOR 2 CHAR'S
       STA 1,S SAVE A COPY
       CLRB
       ASRA GET HI ORDER NIBBLE TO LOWER FOUR BITS
       ASRA
       ASRA
       ASRA
MAK0   CMPA #$0A BIGGER THAN 9?
       BGE ALFA
       ADDA #$30 ADD ASCII BIAS
       BRA MAK1
ALFA   ADDA #$37 FOR A THRU F
MAK1   STA B,S
       TSTB LOOP COUNTER
       BNE EXIT
       INC B
       LDA 1,S
       ANDA #$0F MASK OFF HI ORDER NIBBLE
       BRA MAK0
EXIT   PULS D
       RTS
* COULD USE PULS D,PC
```

The subroutine is intended to show the technique of using the system stack for local variables, and I make no claim about its being the most efficient way to do the job. There are several advantages to writing subroutines this way. First of all, the subroutine is self contained. You don't need to remember to reserve memory for its variables when you include it in a program. Second, the variables only take up space while the subroutine is executing. There is no "permanent" allocation of memory for the variable. In a large program with numerous subroutines, this can reduce variable storage space considerably. Third, and the point of this discussion, the code is position independent since the variables are accessed relative to the stack pointer. Fourth, since the variable is created and destroyed all within the subroutine, you can use the system stack without having to worry about complications arising from the return address being on the stack. Note, however, that if you allocate variables as shown here and then jump to another subroutine, any accesses in that subroutine must have 2 added to the offset value to account for the return address from that subroutine.

One fact that we have not mentioned previously in this discussion, which might be obvious to seasoned programmers and obscure to beginners, is that the above method of creating local variables

every time a subroutine is called, makes it possible for the subroutine to be re-entrant and recursive. That is, it may change the value of a variable and call itself from within until the value reaches some limit (0 for example) then go on to complete all the partially executed recursive calls. One could write a very simple routine to calculate the FACTORIAL function of a number, for example using this sort of variable allocation.

Global Variables on the Stack

Now we have a nice way to create temporary variables on the system stack within a subroutine. How would we go about creating so called Global variables (variables that are defined outside of subroutines and are available to the whole program, and for as long as the program is running)? The following program fragment shows how to create a couple of variables and point the User stack at them, and then how to reference them within a program. By using meaningful EQU statements and a couple of Macros, an assembler program can begin to look like a higher level language program in terms of defining variables.

```
 NAM STKVAR
 TTL STACK VARIABLES DEMO
 OPT PAG,EXP
 PAG
*
* DEMONSTRATION OF LOCAL VARIABLES ON STACK
*
WARMS EQU $CD03
INDEC EQU $CD48
OUTDEC EQU $CD39
PSTRNG EQU $CD1E
PCRLF EQU $CD24
NXTCH EQU $CD27
PUTCHR EQU $CD18
*
BYTE EQU 1
INT EQU 2
*
* MACRO INITIALIZES COUNT, A VARIABLE USED ONLY AT ASSEMBLY TIME
*
INZARG MACRO
COUNT SET 0
 ENDM
*
* DEFINE IS USED TO DEFINE VARIABLES ON THE STACK
*
DEFINE MACRO
&1 SET COUNT  DEFINE VARIABLE OFFSET ON STACK
COUNT SET COUNT+&2  BUMP COUNT FOR NEXT VARIABLE
 ENDM
*
* FIRST DEFINE VARIABLES TO BE USED IN PROGRAM
*
START INZARG
 DEFINE NUMBER,INT
 DEFINE CHAR,BYTE
*
 LEAS -COUNT,S MAKE ROOM FOR VARIABLES
 TFR S,U TRANSFER POINTER TO USER STACK
* DEMO STATEMENTS
 LDA #'a
 STA CHAR,U
 LDD #12345  DECIMAL NUMBER
 STD NUMBER,U
 LDA #$00 CR
 JSR PUTCHR
 LDA #$0A LF
 JSR PUTCHR
 LDA CHAR,U
 JSR PUTCHR SHOULD PRINT a TO TERMINAL
 JMP WARMS
 END START
```

This program assembles with the TSC MacroAssembler and runs. Of course all it does is to do a CRLF and print a small "a" on the screen. The code does prove that it stored the "a" in the variable at CHAR,U and retrieved it successfully after using ACCA for several other instructions. In a program running under FLEX, you would probably leave the stack right where FLEX put it at $C080, but in other environments you would likely want to LDS #$XXXX in order to put it in a known place, since if your program were large and had a large number of variables it might back up past $C000 and overwrite any special drivers that your system uses just below $C000. A good place to put the stack in any program that you write, is just before the current MEMEND value.

```
MEMEND EQU $CC2B
```

would go in the EQUates section of your program, since that is the location of the FLEX MEMEND value.

```
LDS MEMEND
```

would take care of putting the stack at the top of available memory. Then the LEAS -COUNT,S backs the stack pointer up to make room for the global variables at the top of memory. TFR S,U points U at the first global variable, and everything is set up. If you have a four byte REAL number math package, you could define:

```
REAL EQU 4
```

along with the definition of CHAR and INT, and you could then define simple variables by name. If you are going to have a table or an array of CHARS containing 20 elements you could do the following:

```
DEFINE NAME,CHAR*20
```

That statement would reserve 20 bytes in the Global variables and the first location would have the label NAME. Now if you have a small number of variables so that the offset on the U stack is less than 128, you could access element N of that array as follows:

```
LDB #NAME
ADDB N
LDA B,U
```

You could index through an array of characters, say, for example a filename that had been put there by another routine and terminated by a null character ($00) as follows:

```
LEAX NAME,U
BSR PLOOP
.
JMP WARMS

* SUBROUTINE PLOOP STRING POINTER IN X
PLOOP LDA ,X+
 BEQ ENDLOP
 JSR PUTCHR
 BRA LOOP
ENDLOP RTS
```

The instruction LEAX NAME,U is what gets X pointing at the first character in the array. From that point on it is just about the same as any other. Note that a null terminated string allows the print loop to be more efficient since you need only put the single instruction BEQ ENDLOP rather than CMPA #$XX first.

Higher Level Languages

I can't resist the temptation to put in my standard pitch for higher level languages here. Look a few lines up where I used the instruction LEAX NAME,U to get X pointing at the start of a string of characters. Suppose I were not thinking clearly and I simply used LDX #NAME. Now X would contain the value of the offset from the User Stack Pointer at which the string starts. If we have a small number of variables, that would result in X pointing at some low memory address, probably a program instruction. LDX NAME would load X with the contents of that same low memory location. One has to be careful when writing assembler code to use the right instruction. Distinctions are precise, and one little mistake can cause the program to fail in disaster. Point the X register at the wrong address and clear 20 bytes, for example, and you might wipe out the very instruction that is executing currently!

The point is that what I have described above is quite the way that some of the higher level language compilers allocate Global and Local variables. Most of the newer compilers written for the 6809 put all variables on the stack. Many, though not all, generate position independent code. The compiler takes charge of the bookkeeping for you and sees that the stack is properly manipulated and the variables allocated.

Many of the modern compilers generate code that runs very nearly as fast as hand coded assembler programs. The penalty paid for this overhead of the compiler handling details for you, is generally that the compiler treats all variables equally. The assembler programmer may see that if he keeps a count in the B accumulator, he can increment it efficiently and quickly. The compiler will treat all the variables as memory locations. INCB will do the job in the assembler program but the compiler will probably generate code something like:

```
LDD COUNT,U
ADDD #1
STD COUNT,U
```

A good compiler would generate such code as above for the instruction COUNT := COUNT+1 in Pascal or COUNT++ in C. Of course any given routine can't use registers for more than a few of the variables, so that the imagined gain is greater than that actually realized by going to assembler code. In most cases, the modern higher level language compilers generate assembler source code as an intermediate step, and you can look at the code at that point and substitute faster assembler code in critical areas. Most all programs spend most of their time executing a very few lines of code in a repeat loop. That is, for example, they execute initialization code once, but somewhere in the code is a loop that is executed thousands of times during a run of the program.

To cite an example, PAT is written in PL/9. When it first started running reasonably reliably, I re-coded some of the key subroutines that ran most of the time, in assembler. Just about 2% of the object code of PAT is coded directly in assembler, but I realized a speed increase of about 5 times by optimizing those routines.

In these days of 640K, 1 Megabyte, 2 Megabytes of memory coming as standard equipment on computers, the old objection that higher level languages generate more code, though generally true, is largely irrelevant. It used to be true that computer hardware was VERY expensive and the cost of a clever programmer who could code efficiently in assembler and squeeze more program in less hardware was justified. Now the software is the largest portion of the cost of doing things with a computer, and it makes sense to use a little more hardware in order to simplify the process of writing and debugging programs.

I am not speaking from a lopsided perspective, I don't think. I have been writing software for computers for just about ten years now. The first four or five of those years, there was no compiled language that was nearly efficient enough to allow its use for the programs I wrote, so all were written in assembler. I spent four months writing and debugging a fairly complex program that had to do some extensive calculations involving vectors. I routinely do software of this complexity in a few days now. Before you have a chance to say "but", let me add that of course I know more about software and programming now than I did then.

Taking that gain into account, I still feel that writing a program in Pascal or C or PL/9 gives me a time advantage of a factor of two to four over writing the same program in assembler. The proponents of FORTH will claim a factor of as much as ten in programming performance for FORTH over Assembler, and I have no reason not to believe them, though I prefer the more "English like" languages for my work.

I have claimed a reduction in the size of a program source listing of about 5 to 1 when switching from Assembler to a higher level language. I have at least one reader who insists that I am exaggerating, and that in some cases a program written in a higher level language will have a source listing just as long as the assembler source. I don't doubt that to be true in some cases, but I can cite a case in which twelve pages of number crunching calculations in assembler could be reduced to twelve lines of PL/9 program! Of course the twelve lines don't include listings of the math package and scientific function library in the PL/9 program, but the assembler listing doesn't include those function routines either. I suppose the conclusion ought to be that depending on the type of program you are writing, the higher level language source listing may be about as long as, or a great deal shorter than the assembler source listing. For the sort of programs that I write, the latter is true.

The brevity of the listing is not the whole story either. In my situation, two programmers are presently involved, and we are considering adding another programmer to our staff. More programmers are only of value if they can understand someone else's code. I contend that though a very experienced assembler programmer can understand someone else's assembler code, it usually takes longer to become familiar with someone else's assembler code than with someone else's code in Pascal or PL/9 (my first choices for "self documentation"), or even "C" which is generally a little harder to follow (perhaps because I am less familiar with it). FORTH programmers claim that they can follow someone else's FORTH program quite easily. I can't speak from experience on that subject.

As I have said repeatedly whenever this discussion comes up, if software performance (execution time) or size is of such great importance as to override the cost considerations, then of course, assembler code is the only way to go.

Disk Controller Troubles

Now that my personal GMX DMA disk controller is back to functioning flawlessly, the system at work started getting flaky. I had tried the old "wiggle the cards" trick a number of times, and it seemed to work fine, but the system seemed to need the cure more often than usual. The other day the system was turned on in the morning and it worked fine. Later in the day I went to use it after it had been sitting idle for a couple of hours and I couldn't read a disk file at all.

I remembered some trouble I had had a long time ago with the SWTPc DMAF controller board. The 1791 disk controller had become heat sensitive. I could run the system for a while and it would then get flaky. I discovered that blowing a fan directly on the 1791 would make it work. Later I discovered that one of the trim adjustments on the board would also make it work after it warmed up. Eventually the problem got so bad that I decided to try a new 1791, and it worked fine. I suspect that one of the ribbon cables got pushed against the 1791 and it got no air circulation and overheated. We are presently waiting for a replacement, and running the system with the cover off and the trim adjustment set for the best compromise between hot and cold operation. I wonder if anyone else has had 1791 troubles of a similar nature?

Ramblings

This column is a little shorter than usual because of my current time pressures. I am involved in the completion of a consulting project, and also preparing for a vacation trip. My wife and I are celebrating our 25th wedding anniversary on October 14. We are definitely not world travelers, but a couple of years ago we had an exchange student from Brazil, and we are going to see him and his parents as our anniversary present. We leave for Brazil on October 13 and return to the U.S. on November 3, so we have been busy making preparations.

I expect I will have a little time to look at computers and their use in Brazil and I will report what I have found when I return.

EOF

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

The Proposed ANSI C Standard

This chapter continues the discussion of the proposed ANSI C standard and the discussion of common problem areas in the use of the C language.

The proposed standard specifies that comments may not be nested. Nested comments are a nonstandard feature of many current compilers. They will probably remain a nonstandard feature of many future C compilers because of their convenience. However, the default action of a conforming compiler must be to disallow nested comments.

One of the primary reasons for disallowing nested comments is directly associated with one of the primary reasons for allowing them. If a trailing comment delimiter (*/) is accidentally dropped or not coded in input to a compiler which disallows nested comments, the compiler can provide better diagnostics and error messages on subsequent leading (/*) and trailing (*/) comment delimiters than in input to a compiler currently allowing nested comments.

The primary reason for allowing nested comments is to allow entire sections of a program to be readily logically deleted without dropping them from the source code entirely. However, this may be better performed by surrounding the code to be deleted with an "#ifdef xxxxxx" - "#endif" pair, where "xxxxxx" is not defined. The only exception to this is when the "#ifdef xxxxxx" - "#endif" pair spans disjoint "#ifdef" - "#endif" pairs, but even this case may be properly handled with a little thought and cleverness, and should be rare.

A preprocessing directive is introduced by an unquoted "#" symbol and terminated by a new-line symbol not preceded by a "\" symbol. The "#" may be preceded by or followed by tabs or spaces, and the terminating new-line symbol may be preceded by tabs or spaces. Preprocessing directives containing only "#", optional tabs and spaces, and new-line symbols, are accepted and ignored.

They are called "preprocessing directives" because they are assumed to occur in a separate compilation stage occurring before the analysis of the syntax of the resulting source program. Comments are also detected and dropped during this pass, but they are not normally referred to as preprocessing directives.

Preprocessing directives of the following forms:

```
#define macro body new-line
#define macro(list) body new-line
```

define a macro name which indicates that subsequent unquoted occurrences of the macro name will be replaced by a processed version of the macro body. A macro name may be redefined only by one of the same type (parameterized or not) and identical body.

In the first (non-parameterized) case of definition of the directive, the body of the macro replaces each occurrence of the macro name without modification. In the second case, the body of the macro is logically modified, with unquoted occurrences of the names in the macro parameter list being replaced by the corresponding strings in each call, before the replacement is performed. Macro parameter list names immediately preceded by a "#" symbol in the macro body are logically surrounded by double-quote symbols during the replacement process. Macro parameter names immediately preceded by a "##" sequence are concatenated to the preceding token.

Macro names are replaced outside of quoted strings and character constants. The replaced sequence is rescanned for additional macro names; however, if a macro name itself is discovered during this rescanning, it is not replaced, in order to prevent macro loops. Since adjacent quoted strings are assumed concatenated in the proposed standard, the use of macro parameters is further enhanced.

For example, given the following declaration:

```
#define debug(s,t) \
   printf("x"#s"=%d,x"#t"=%s", \
     x##s,x##t);
```

the following text:

```
debug(1,2);
```

would be equivalent to the following text:

```
printf("x""1""=%d,x""2""=%s",x1,x2);
```

which is equivalent to the following text:

```
printf("x1=%d,x2=%s",x1,x2);
```

The directive may be continued across multiple source lines by preceding the intermediate new-lines with "\" symbols.

The definition of a macro name may be removed with a preprocessing directive of the following form:

```
#undef macro new-line
```

for redefinition or for other purposes, such as conditional inclusion and exclusion of program text.

Source lines may be included from other files with preprocessing directives of the following forms:

```
#include "filename" new-line
#include <filename> new-line
#include macro-name new-line
```

in which the first form searches the current directory before searching a common directory, the second reverses the process, and the third may represent and evaluate to either of the preceding forms. Source lines included by this process are processed almost identically to source lines included in the original source file. Nesting of this process is allowed to an implementation-defined limit.

The definition or nondefinition of a macro name may be detected with preprocessing directives of the following forms:

```
#ifdef macro-name new-line
#ifndef macro-name new-line
```

Source lines are included or excluded until one of the following corresponding preprocessor directives is encountered:

```
#else
#endif
```

and, if the "#else" directive is encountered, the inclusion or exclusion of source lines is reversed until a corresponding "#endif" directive is encountered. These preprocessor structures may be nested to some implementation-defined limit.

A similar (but more powerful) preprocessing directive which allows the inclusion or exclusion of lines of source code, has the following forms:

```
#if constant-expression
#if defined macro-name
#if defined(macro-name)
#if !defined macro-name
#if !defined(macro-name)
```

and uses the same interpretations of the "#else" and "#endif" directives. In addition, the "#elif" directive is equivalent to an "#else" and an "#if" directive.

This constant-expression may involve any previously-defined or self-defined constants and arithmetic or logical operators, but may not involve the sizeof or cast operations, or enumerated constants. Computations are performed in the equivalent of data type "long int".

Actually, "defined" may appear in any constant-expression used in an "#if" directive. It evaluates to 1 (true) if macro-name has been defined by an "#ifdef" directive and not more recently undefined by an "#undef" directive. The second and third directives above are equivalent to an "#ifdef" directive and the fourth and fifth directives above are equivalent to an "#ifndef" directive.

Another directive, which is seldom used by humans in writing C programs, but is often used by programs which write other C programs, has the following forms:

```
#line line-number
#line line-number file-name
```

The line number of a source line in a C program is defined as one more than the number of new-lines preceding it in its source file. The "#line" directive allows the line number (and, optionally, the file name) to be overridden to assist the programmer in finding syntax and logic errors. The predefined macro names "__LINE__" and "__FILE__" represent the line number and source file name of the current source file, possibly overridden by "#line" directives.

The proposed standard provides an implementation-dependent preprocessor directive of the following form:

#pragma character-sequence

The character-sequence is processed in an implementation-defined manner. If a given conforming implementation does not recognize a given "#pragma" character-sequence, it ignores it silently.

C PROBLEM

Difficulties caused by alignment and data type lengths are probably one of the most troublesome traits of the C language. Neither the K & R book nor the proposed ANSI C standard address either type of problem, as both leave alignment and data type length considerations to the implementation. Such problems normally appear during initial program debugging or during maintenance modification; however, they may also appear during porting to new compilers, operating systems, or computers. These problems may be quite subtle, as no syntax errors are normally generated by alignment or data type length mis-matches.

Given the following structure declaration:

```
struct s1
{
    char c1;
    short int s1;
    int i1;
    long l1;
    float f1;
    double d1;
}   st1;
```

what are the possible values for "sizeof(st1)"? Consider alignment and data type length separately. Also consider the effects on both for computers with byte lengths of other than eight bits and word lengths of other than eight, sixteen, or thirty-two bits. Determine the arrangement of declarations which would provide the minimum value of "sizeof(st1)".

Given the same structure declaration as above, consider the effect of moving the following declaration:

```
char *p1;
```

through the structure, starting with placing it before the first declaration and ending with placing it after the last declaration, assuming the same conditions.

This is not the extent of the complexity of the situation. On some combinations of compilers and computers, there are multiple pointer lengths for on-segment and off-segment references (e.g. near and far pointer types in Microsoft C for the IBM PC) or for different data types.

EXAMPLE C PROGRAM

Following is this month's example C program; it is the first part of an internal B+ tree manipulation program. It is long but instructive. *Rather than entering it manually, it may be downloaded from the computer bulletin board with telephone number 404-493-4708.*

```
/* B+ tree search, insertion and deletion */
#define N 2
#define NN 4  /* index page size */
#define L 2
#define LL 4  /* data page size */
#define NULL 0
#define LEAF 0
#define INDEX 1
typedef struct page
{
   int    page_type;
   union
   {
      struct
      {
         int    m;
         struct page *p0;
         struct
         {
            int    key;
            struct page *p;
         }
         e[1+NN];
      }
      indexp;
      struct
      {
         int    k;
         struct
         {
            int    key;
            int    count;
         }
         d[1+LL];
      }
      leafp;
   }
   type;
}
PAGE, *REF;

typedef struct item
{
   int    key;
   struct page *p;
}
ITEM;

typedef struct info
{
   int    key;
   int    count;
}
INFO;
REF root;

main()
{
   REF q, leaf;
   int    x, h;
   ITEM u;
   char   *new();

   scanf("%d", &x);
   root = (REF)new(sizeof(*root));
   root->page_type = INDEX;
   root->type.indexp.m = 1;
   root->type.indexp.p0 = leaf =
      (REF)new(sizeof(*leaf));
   leaf->page_type = LEAF;
   leaf->type.leafp.k = 0;
   root->type.indexp.e[1].key = x;
   root->type.indexp.e[1].p = leaf =
      (REF)new(sizeof(*leaf));
   leaf->page_type = LEAF;
   leaf->type.leafp.d[1].key = x;
   leaf->type.leafp.k = 1;
   leaf->type.leafp.d[1].count = 0;
   while (x != 0)
   {
      printf("search key %d\n", x);
      search(x, root, &h, &u);
      if (h)
      {  /* insert new base page */
```

```
            q = root;
            root = (REF)new(sizeof(*root));
            root->page_type = INDEX;
            root->type.indexp.m = 1;
            root->type.indexp.p0 = q;
            root->type.indexp.e[1].key = u.key;
            root->type.indexp.e[1].p = u.p;
        }
        printtree(root, 1);
        scanf("%d", &x);
    }
    scanf("%d", &x);
    while (x != 0)
    {
        printf("delete key %d\n", x);
        delete(x, root, &h);
        if (h)
        {  /* base page size was reduced */

            if (root->type.indexp.m == 0)
              {
                  q = root;
                  root = q->type.indexp.p0;
              }
          }
          printtree(root, 1);
          scanf("%d", &x);
      }
    }


    char    *new(n)
    unsigned int    n;
    {
        char    *p, *malloc();

        if ((p = malloc(n)) == NULL)
        { /* no space available */
            printf("no space at all\n");
            exit(1);
        }
        else
            return(p);
    }


    /* Search key x on B_tree with root a; if found, increment
counter, otherwise insert an item with key x and count 1 in
tree. If an item emerges to be passed to a lower level, then
assign it to v, h = "tree a has become higher" */
    search(x, a, h, v)
    int     x;
    REF a;
    int     *h;
    ITEM *v;
    {
        int     k, l, r;
        REF q;
        ITEM u;
        INFO z;
        char    *new();

    /* search for key x on page *a; h = false */

    if (a->page_type == LEAF)
    {
        l = 1;
        r = a->type.leafp.k;
        while (r >= l)
        {  /* binary array search */
            k = (l + r) / 2;
            if (x <= a->type.leafp.d[k].key)
                r = k - 1;
            if (x >= a->type.leafp.d[k].key)
                l = k + 1;
        };
        if (l - r > 1)
        {  /* found */
            a->type.leafp.d[k].count += 1;
            *h = 0;
        }
        else
        {
            z.key = x;
            z.count = 1;
            insert1(a, r, h, v, &z);
        }
    }
    else
    {
        l = 1;
        r = a->type.indexp.m;
        while (r >= l)
        {  /* binary array search */
            k = (l + r) / 2;
            if (x <= a->type.indexp.e[k].key)
                r = k - 1;
            if (x >= a->type.indexp.e[k].key)
                l = k + 1;
        };
        if (l - r > 1 )
        { /* found */
            q = a->type.indexp.e[k].p;
            search(x, q, h, &u);
        }
        else
         {  /* item is not in this page */
            if (r == 0)
                q = a->type.indexp.p0;
            else
                q = a->type.indexp.e[r].p;
            search(x, q, h, &u);
            if (*h)
            { /* an item u is being passed up */
                insert1(a, r, h, v, &u);
            }
        }
    }
}
```

EOF

Basically OS-9

The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL
60139

DOES THIS REGISTER WITH YOU?

Ever come across a programming problem that just doesn't make sense? You look at the code, over and over. You've used the same code, or at least something similar before and it's always worked. This following example happened to me. See if you can spot the mistake. I passed two parameters from Basic09 to a machine language subroutine. The machine language program was to set the priority of a process. Registers A and B were to be loaded with process ID and priority, respectively. The call from Basic09 could pass either integer or byte values. If a byte was passed, all that was necessary was to load it into a register. If an integer (two bytes) were passed, the second half of it would be used. Here is the codes with only the essentials.

```
      LDD SIZE1,S
        CMPD #2
        BNE S001
        LDA [PAR1+1,S]
        BRA S002
       S001 LDA [PAR1,S]
       S002 EQU *
        LDD SIZE2,S
        CMPD #2
        BNE S003
        LDA [PAR2+1,S]
        BRA S003
       S001 LDB [PAR2,S]
       S002 EQU *
        OS9 F$SPRIOR
```

The parameters are passed on the stack. SIZE1 and SIZE2 indicate whether PAR1 and PAR2 point to an integer or byte value. When I ran the assembled code, it would always return with an error #238--Bad Process ID. Everything looked fine, but it would not work. Do you see the mistake?

Time's up! I sure the sharp programmers out there spotted it right away. The logic is correct, but the registers were used incorrectly. Registers A and B are the D register. Therefore, the line LDD SIZE2,S wrote over the process ID which was in register A. The value could have been saved using PSHS A and retrieved later with PULS A. But a better solution would be to use another register. In this case, I wasn't using the X register, so I used LDX and CMPX. Then everything worked out fine. Another benefit was the code ended up two bytes smaller.

Using PSHS and PULS is very useful in saving registers. If you find yourself writing assembly language subroutines, you may want to save any or all of the registers on entry and restore them when leaving the subroutine. The overhead is two bytes to PSHS registers and two bytes to restore them. This can save considerable heartache later. PSHS and PULS are represented by a byte, each. A post-byte is added for the registers. Each bit in the post byte represents a register.

```
Bit No.   7  6  5  4  3  2  1  0
Reg.     PC  S  Y  X DP  B  A CC
```

The push order is from left to right. The pull order is from right left. Notice the program counter, PC, goes on first, but comes off last. This makes it useful for subroutines. Let us say you pushed the X and A registers at the start of the routine. As a last step you can use:

```
PULS A,X,PC
```

Pulling PC restores the program counter and returns execution to wherever the call originated. As another note, the U register can be used as a stack pointer. Use PULU and PSHU. Just remember to point it to some data memory area.

But for OS-9 the U register has a greater importance, as do all the other registers. Whenever a process is created, a data area is created for it in memory. Registers, U and DP, point to the start of data memory. X and S point to the division between the data area and parameter area. And Y points to the end of the parameter area. Register D contains the size of the parameter area. And PC, the program counter, is loaded with the entry point of the module. Mapping this in memory, it appears:

```
U, DP >  ---------------------- low mem

           data are

X, S >   ----------------------

         parameter area

Y    >   ---------------------- high mem

         D = parameter area size
         PC = absolute address of entry
```

Such a set up can make programming extremely easy. Direct page addressing can be used or indexed addressing can be used. For the first 256 bytes, the direct method works well, since DP points to the most significant byte. Entering

```
LDA <2
```

will load register A with the 3rd byte of the data area. The left arrow tells the assembler to use direct addressing. The 2 makes it the 3rd byte with 0 and 1 being the first and second.

This line could also be written:

```
LDA 2,U
```

using the indexed addressing mode. Both will do the same thing. Index addressing lets you go further than the 256 byte limit of direct addressing.

The X register ends up pointing to the parameter list. That is rather convenient if you consider that many of the OS-9 calls use the x register to point to things like pathlists, filenames, and module names. Let us say you want to open a path to a file for reading. On entry the X register will point to some parameter list. So entering the line:

```
READ FILENAME
```

will result with the OS-9 SHELL executing the module READ and the X register pointing to "FILENAME". The code to handle this could simply be:

```
lda #1
os9 F$Open
bcs Error
```

After the call, the X register is advanced. So, multiple file names could be easily handled. The X register sometimes is used to point to buffer areas. The most common ones are the read and write calls.

Now for the stack pointer. The stack is perhaps the most important register of them all. I recently wrote a little program and forgot to supply a stack area. Wrong! Everything crashed. The moral is to supply an adequate stack area. Remember every procedure has its own stack area. The stack builds upward as viewed from my diagram. If you don't supply enough area for it, it will build into the data area or worse. It may go to places, it shouldn't.

Besides saving and restoring registers as I pointed out earlier, the stack can be used for other things. Let us say you need instant memory. Maybe it is subprogram for Basic09. You want to create two integers, storing 0 in the first and 3 in the second. The following code would handle this.

```
leas -4,s
clra
clrb
std 0,s
ldd #3
std 2,s
```

The leas -4,s moves the stack up (towards low mem) by 4 bytes. Now to that everything must be referenced from the stack pointer's new location and when your all done restore the stack pointer with a leas 4,s. This technique is used for 'auto' variables in C Language programs. Another thing, the stack pointer can offer throw away memory. Testing the X register for 0 can be done with:

```
cmpx #$0000
```

which will take 3 bytes. The same thing can be done with:

```
stx -2,s
```

This stores the value of X two bytes ahead of the stack pointer, a throw away area. But it sets the appropriate condition code registers and it takes only 2 bytes.

The A and B registers have their own responsibility in OS-9. In all the system calls that involve input/output, the A register is the path number. Occasionaly it holds other information like language/type code for some of the other calls, but predominantly it is used for path number. The B register is used for a number of things like attributes/revision level, function code for GetStt and SetStt, and directory attributes. But its most important role is for error codes. If the carry bit is set in the condition code register, B will contain the error code. With only a few exceptions, all the calls return their error status in this way.

I've already mentioned the condition code register CC. Its importance to OS-9 is to indicate an error. If the carry bit, bit #0, is set then an error is indicated. If it is cleared, no error. In the past, I used to set and reset the bit with calls like:

```
orcc #%00000001
andcc #%11111110
```

Each of these codes generate 2 bytes. Now I use:

```
coma
clrb
```

The 'coma' complements the contents of register A. This in not important, but the carry bit is always set. The 'clrb' clears the B register and sets the carry bit. Each line generates only 1 byte of code. Another savings!

I haven't mentioned much about the Y register. It does point to the end of the parameter area. This may or may not be important, depending on what you are doing. I usually try to use the X register for indexing where it is possible. The Y register operands involve more bytes of code than the X register. The Y register code is the same as the X with a $10 tacked on the front. This means, everytime you use the Y register where the X would have worked, you add an extra byte of code. I like to think of the Y register as an stand-by register. It does get used in some system calls like FCRC, IREAD and I$WRITE. Usually it is used for a counter.

The registers used in OS-9 have a great importance. There is so much that I haven't covered here. If you aren't into assembly language programming, you may feel this isn't very important. But even the higher level language use the registers extensively. In Basic09, parameter pointers are passed on the stack with their size. In C language, the Y register is used to point to the data area instead of U. The stack is a place where auto variable are created. And it is used for pass values to the functions. So, even if you work with these languages, remember the registers are at the center of things.

THE KO PUNCH, 1-2-3 POW!

A few months back I started talking about KBASIC. Kbasic is a basic compiler from Lloyd's I/O. It works in 3 stages. There is a token file generator, KS. KA converts the token file to assembly language source code. Finally, KO creates object code from the source code. KO is the surprise of the bunch. It is a sharp assembler that can be used for many of your assembly language projects.

Usually, I use the Microware Interactive Assembler. I have no complaints about it. It handles most of my programming needs. But KO does a nice job too, with a few extras. I won't go into details about it as an assembler, since most of you are familiar with them. But let me tell you a little about its added features.

It handles IF...ENDC a little differently. With MIA you could create a line like:

```
STATUS SET 0
  IFEQ
  CLRB
  ELSE
  LDB #1
  ENDC
```

Now if STATUS is 0, we clear the contents of register B. Change it to another number and B is loaded with a 1. With KO the same thing could be written:

```
STATUS SET 0
  IF STATUS=0
  CLRB
  ELSE
  LDB #1
  ENDC
```

Here STATUS is compared to 0. We could have also written the second part as:

```
IF STATUS=1
LDB #1
ENDC
```

A whole range of possibilities is here, since the assembler recognizes the usual operators, like +, =, < and ! (not).

Another feature is, it recognizes mnemonics that were features of the 6800. Some familiar ones are DEX, INX, TAB, and CLC. KO replaces these codes the actual 6809 codes. So, DEX becomes LEAX -1,X and TAB is TFR A,B. The code to return with a Bad Path Number error could be written:

```
LDB #E$BPNUM
SEC
```

The assembler would replace the SEC with the appropriate code, ORCC #$01. There are 27 mnuemonics in all. They provide an alternate mode of programming.

Finally, KO allows the use of local and global labels. Labels appear in the first column, the label field. They refer to a place in memory where something is stored or an entry point of the program occurs. Occasionally, they are set equal to some value with the SET or EQU opcode. If a label appears more than once (with the exception of SET) in the label field, an error of multiply defined labels occurs. Now imagine you wish to create a library of subroutines that can be merged with you assembly language programs. Extreme care would have to be taken to insure that no labels get used more than once. KO removes this problem. It allows using local labels.

```
  SLL
XFER EQU *
 LDB #10
_LOOP LDA ,X+
 STA ,Y+
 DECB
 BNE _LOOP
 RTS
 ELL
```

S LL tells the assembler that everything starting with the underscore, "_", is a local label, here it is _LOOP. The ELL tells the end of the local labels. The scope of local labels exist between the SLL and ELL. The label XFER is global. It is recognized to the outside. If it had started the the underscore, it would have been a local label too. In another part of code, bounded by SLL and ELL, the same local label can be reused.

I have found that these little features make KO a sharp little assembler. Even when I am not programming with KBASIC, I keep the assembler in my commands directory. More and more I am giving my programs the KO punch.

IT FINALLY ARRIVED

At long last it is here. Radio Shack has finally produced the Color Computer III. By the time you read this, many of you will probably be looking for it under your Christmas trees. The new machine is a 128K color computer, expandable to 512K. It has 64 colors and can support screens up to 640 X 192 resolution. And best of all the new machine will run on OS-9 Level II.

That is it for this month. Next month we'll bring you more on OS-9. Take care!

EOF

"CRYPTOQUOTE"

By: Mickey E. Ferguson
POB 87
Kingston Springs, TN 37082

I must be getting old. My wife says not old, just lazy. *(Editor's Note: probably both!)* You see, I developed the program described here by the method I have always referred to as "Ron Anderson's Lazy Programming Method". I have always programmed by what has come to be called the *Hacker's Creed* where every byte and every machine cycle counted. But this program is a bit different; it is the first program that I have ever written where I was more interested in using it than in writing it.

For many years I have been hooked on the puzzles in the daily newspapers. And an unworked crossword puzzle is an open challenge! But the cryptoquote has always been my favorite. Some people judge the daily newspapers in their town by the editorial content, or the coverage of local (or state or national or world) news. Others consider the sports coverage or other things that I haven't even thought about. For me, the only daily newspaper worth actually paying money for is the one with the cryptoquote in it!

As newspapers tend to use puzzles as fillers, they tend to be stuffed into whatever space is available and are often rather tiny. I used to copy the cryptoquote from the newspaper onto notebook paper and work from that. When we got our first computer (an SWTPC 6800) I wanted to write a program for it that would allow me to use the CRT as my worksheet for cryptoquote puzzles. But I was not skilled enough to write it in assembly language; so I had to wait until we aquired a suitable version of BASIC. When SWTPC's 8k BASIC arrived, I wrote the program in BASIC. But it ran soooooo slooooooowly that I gave up on the idea for a time. When TSC's Extended Basic came along, it was much faster and finally made my cryptoquote program in BASIC a viable proposition. The listing in figure 1 is the xBASIC program.

But the xBASIC program still ran too slowly for my tastes and I happened to have Microware's A/BASIC Compiler; so I converted it from xBASIC to A/BASIC and compiled it. The listing in Figure 2 is the A/BASIC program. Now A/BASIC probably generates the fastest machine code of any BASIC compiler ever written for the 68xx based machines (sure wish I had the source to A/BASIC cause I think I could make it faster); but the A/BASIC program still ran too slowly for my tastes.

The hacker in me took over and I disassembled the machine code the A/BASIC program had generated with the objective of optomizing it for speed. This led to the listing in Figure 3. Now I am not saying that the program in Figure 3 is the ultimate optomization of the code that A/BASIC had generated; but it is a good deal faster and about 750 bytes shorter. About the only comments in Figure 3 are the lines of A/BASIC source that caused the assembler code to be generated. There are several areas where the program in Figure 3 could be improved but, as I said before, I was more interested in using the program than in writing it.

All three programs function identically when run. The CRT screen is cleared [PRINT CHR$(12)], and you are prompted to begin inputting data from the cryptoquote in the newspaper. All printable ASCII characters are allowed as input (except in the xBASIC program). Input is terminated with a null line, i.e. a carrage return only. The screen is then cleared and the cryptoquote is then displayed with a space between each letter of a word, three spaces between words, and two blank lines between each line of the encrypted quote. A maximum of ten lines of fourty characters is allowed which is more than enough to handle any cryptoquote I have seen. You are then prompted for a command. The command can have one of three formats, x=y, RESET, or STOP. x and y may be any printable ASCII character and all occurances of x will have y displayed on the line directly above it in the encrypted text. RESET removes all changes you have made and displays the encrypted quote with two blank lines between each line of text. In other words, it restores you to the point then you stopped inputting the quote. STOP exits the program and returns you to FLEX.

I felt for sometime that the only true hackers left were the readers of 68 Micro Journal. And now I understand that many of the newer readers are relative beginners who have CoCo's and are wanting to learn more about programming it. If you fall into the latter catagory, you may wish to take a good close look at the listing in Figure 3. This will give you a good idea of the code the compiler generates in response to BASIC source code. You will see the code generated by each line of BASIC source is a complete module, not depending on the line before it to preserve the contents of any registers. You will also see things like how the compiler sets up a FOR loop, and how it deals with the NEXT at the end of the loop. Things like these you will find useful in writing your own programs in assembly language. There are many subroutines in the run-time package part which you will also find very useful. For example, the subroutine beginning at line 778 determines the length of a string [the LEN(X$) function], the subroutine beginning at line 729 prepares a two byte binary value for printing as decimal, and the subroutine beginning at line 647 is a buffered input routine allowing for backspacing and line cancel.

I still have one little problem. The program still runs tooo slooooooly for me. I wonder if I could convince Foxy (my wife) to let me buy a Mustang system for my cryptoquote program.

Cryptoquote is a trademark of King Features Syndicate, ABASIC is a trademark of Microware Corp.

figure 1.

```
10 DIM B$(10),C$(10)
20 PRINT CHR$(12)
30 LET X%=0
40 LET X%=X%+1
50 PRINT "LINE";X%;
60 INPUT C$(X%)
70 IF C$(X%)="" THEN 130
80 GOSUB 520
90 FOR Y%=1 TO LEN(C$(X%))
100 LET B$(X%)=B$(X%)+" "
110 NEXT Y%
120 GOTO 40
130 LET Z%=X%-1
140 PRINT CHR$(12)
150 FOR X%=1 TO Z%
160 PRINT B$(X%)
170 PRINT C$(X%)
180 PRINT
190 NEXT X%
200 PRINT
210 INPUT D$
220 IF D$="STOP" THEN 600
230 IF D$="RESET" THEN 440
240 IF D$="PRINT" THEN 400
250 IF D$="CRT" THEN 420
260 IF MID$(D$,2,1)<>"=" THEN 380
270 FOR X%=1 TO Z%
280 LET E$=""
290 FOR Y%=1 TO LEN(C$(X%))
300 IF MID$(C$(X%),Y%,1)<>LEFT$(D$,1) THEN 330
310 LET E$=E$+MID$(D$,3,1)
320 GOTO 340
330 LET E$=E$+MID$(B$(X%),Y%,1)
340 NEXT Y%
350 LET B$(X%)=E$
360 NEXT X%
370 GOTO 140
380 PRINT CHR$(12);"WHAT";CHR$(7)
390 GOTO 150
400 EXEC,"PORT=3"
410 GOTO 140
420 EXEC,"PORT=1"
430 GOTO 140
440 FOR X%=1 TO Z%
450 LET E$=""
460 FOR Y%=1 TO LEN(C$(X%))
470 LET E$=E$+" "
480 NEXT Y%
490 LET B$(X%)=E$
500 NEXT X%
510 GOTO 140
520 LET D$=""
530 FOR W%=1 TO LEN(C$(X%))
540 LET D$=D$+MID$(C$(X%),W%,1)
550 LET D$=D$+" "
560 IF MID$(D$,LEN(D$)-1,1)=" " THEN LET D$=D$+" "
570 NEXT W%
580 LET C$(X%)=D$
590 RETURN
600 END
```

figure 2.

```
05 OPT H,S
10 DIM X(1),Y(1),Z(1)
   DIM D$(1,80),E$(1,80),B$(10,80),C$(10,80)
   STACK=$0FFF
20 PRINT CHR$(12);
   FOR X=1 TO 10
   LET B$(X)=""
   LET C$(X)=""
   NEXT X
   LET D$=""
   LET E$=""
30 LET X=0
   LET Y=X
   LET Z=X
40 LET X=X+1
   IF X>10 THEN 130
50 PRINT "LINE";X;
60 INPUT BUF$
70 IF BUF$="" THEN 130
   LET C$(X)=BUF$
80 GOSUB 520
90 FOR Y=1 TO LEN(C$(X))
100 LET B$(X)=B$(X)+" "
110 NEXT Y
120 GOTO 40
130 LET Z=X-1
140 PRINT CHR$(12);
150 FOR X=1 TO Z
160 PRINT B$(X)
170 PRINT C$(X)
180 PRINT
190 NEXT X
200 PRINT
210 INPUT BUF$
220 IF BUF$="STOP" THEN 600
230 IF BUF$="RESET" THEN 440
    LET D$=BUF$
260 IF MID$(D$,2,1)<>"=" THEN 380
270 FOR X=1 TO Z
280 LET E$=""
290 FOR Y=1 TO LEN(C$(X))
300 IF MID$(C$(X),Y,1)<>LEFT$(D$,1) THEN 330
310 LET E$=E$+MID$(D$,3,1)
320 GOTO 340
330 LET E$=E$+MID$(B$(X),Y,1)
340 NEXT Y
350 LET B$(X)=E$
360 NEXT X
370 GOTO 140
380 PRINT CHR$(12);"WHAT";CHR$(7)
390 GOTO 150
440 FOR X=1 TO Z
450 LET E$=""
460 FOR Y=1 TO LEN(C$(X))
470 LET E$=E$+" "
480 NEXT Y
490 LET B$(X)=E$
500 NEXT X
510 GOTO 140
520 LET D$=""
530 FOR Y=1 TO LEN(C$(X))
540 LET D$=D$+MID$(C$(X),Y,1)
550 LET D$=D$+" "
```

```
560 IF MID$(D$,LEN(D$)-1,1)<>" " THEN 570
565 LET D$=D$+" "
570 NEXT Y
580 LET C$(X)=D$
590 RETURN
600 STOP
610 END
```

figure 3.

```
                            NAM   QUOTE.BIN

                    0050    MAXLEN EQU  80

                          * STANDARD PRE-NAMED LABEL EQUATES

                    CC00    BS_CHR EQU  $CC00
                    CC01    DL_CHR EQU  $CC01
                    CD03    WARMS  EQU  $CD03
                    CD15    GETCHR EQU  $CD15
                    CD18    PUTCHR EQU  $CD18

0000                        ORG    $0000

0000                NEXT_X  RMB    2
0002                NEXT_Y  RMB    2
0004                X_TEMP  RMB    2
0006                BUFPNT  RMB    2

                          * 0010 DIM X(1),Y(1),Z(1)

0008                X_VAR   RMB    2
000A                Y_VAR   RMB    2
000C                Z_VAR   RMB    2

                          * DIM D$(1,80),E$(1,80),B$(10,80),C$(10,80

000E                D_STR   RMB    MAXLEN
005E                E_STR   RMB    MAXLEN
00AE                B_STR   RMB    10*MAXLEN
03CE                C_STR   RMB    10*MAXLEN
06EE                IO_BUF  RMB    $81
076F                ST_BUF  RMB    $D0

1000                        ORG    $1000

                    *       STACK=$0FFF

1000 10CE 0FFF      QUOTE   LDS    #QUOTE-1

                          * 0020 PRINT CHR$(12);

1004 86   0C                LDA    #12
1006 BD   13EC              JSR    OUTEEE

                          * FOR X=1 TO 10

1009 CC   0001              LDD    #1
100C DD   08                STD    X_VAR
100E C6   0A                LDB    #10
1010 DD   00                STD    NEXT_X

                          * LET B$(X)=""

1012 D6   09        FOR_LP  LDB    X_VAR+1
1014 86   50                LDA    #MAXLEN
1016 3D                     MUL
1017 C3   005E              ADDD   #B_STR-MAXLEN
101A B0   13D0              JSR    STNG_0

                          * LET C$(X)=""

101D D6   09                LDB    X_VAR+1
101F 86   50                LDA    #MAXLEN
1021 3D                     MUL
1022 C3   037E              ADDD   #C_STR-MAXLEN
1025 BD   13D0              JSR    STNG_0

                          * NEXT X

1028 BD   13BC              JSR    NEXTX
102B 2D   E5                BLT    FOR_LP
102D 26   03                BNE    LPEXIT
102F 50                     TSTB
1030 27   E0                BEQ    FOR_LP

                          * LET D$=""

1032 8E   000E      LPEXIT  LDX    #D_STR
1035 BD   13DC              JSR    STNG20

                          * LET E$=""

1038 8E   005E              LDX    #E_STR
103B B0   13DC              JSR    STNG20

                          * 0030 LET X=0

103E CC   0000              LDD    #0
1041 DD   08                STD    X_VAR

                          * LET Y=X

1043 DD   0A                STD    Y_VAR

                          * LET Z=X

1045 DD   0C                STD    Z_VAR

                          * 0040 LET X=X+1

1047 9E   08        LN__40  LDX    X_VAR
1049 30   01                LEAX   1,X
104B 9F   08                STX    X_VAR

                          *       IF X>10 THEN 130

104D DC   08                LDD    X_VAR
104F 1083 000B              CMPD   #11
1053 24   68                BHS    LN_130

                          * 0050 PRINT "LINE";X;

1055 B0   1442              JSR    SETBUF
1058 8E   1566              LDX    #LINE
105B B0   1477              JSR    MOVSTR
105E DC   08                LDD    X_VAR
1060 BD   1485              JSR    CVTVAR
1063 B0   1448              JSR    PSTRNG

                          * 0060 INPUT BUF$

1066 BD   13EF              JSR    INPUT

                          * 0070 IF BUF$="" THEN 130

1069 8E   06EE              LDX    #IO_BUF
106C BD   14E1              JSR    LEN
106F 5D                     TSTB
1070 27   4B                BEQ    LN_130

                          *       LET C$(X)=BUF$

1072 D6   09                LDB    X_VAR+1
1074 86   50                LDA    #MAXLEN
1076 30                     MUL
1077 C3   037E              ADDD   #C_STR-MAXLEN
107A 8E   06EE              LDX    #IO_BUF
107D BD   1303              JSR    STNG_1

                          * 0080 GOSUB 520

1080 BD   12C5              JSR    LN_520

                          * 0090 FOR Y=1 TO LEN(C$(X))

1083 BD   1391              JSR    Y_CUX

                          * 0100 LET B$(X)=B$(X)+" "

1086 D6   09        LN_100  LDB    X_VAR+1
1088 86   50                LDA    #MAXLEN
108A 3D                     MUL
108B C3   005E              ADDD   #B_STR-MAXLEN
108E 34   06                PSHS   D
1090 D6   09                LDB    X_VAR+1
1092 86   50                LDA    #MAXLEN
```

To Be Continued Next Month

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01543

C LIBRARY ROUTINE CONVERSION to FORTH

There are a lot of very good programs available in the public domain written in C, but I prefer to work in FORTH. Therefore, if I am going to make use of the programs, *I must first convert them to FORTH*. As one step in that process, I have written a series of routines which duplicate the action of C library routines for working with ASCII strings. Some of these routines are duplicated here.

The listing called CHARTEST.FTH is in FLEX text file format to make it easier to use in creating programs. If you prefer to work with the traditional FORTH screen format, then I recommend that you put each routine in a separate screen.

As you can see from an examination of the CHARTEST.FTH listing, the routines are essentially self-documenting. The routines are not necessarily as short as they could be, but each one has been as thoroughly tested as practical, and there should be no surprises.

Each routine is entered with the character to be tested on top of the Data Stack. The routine replaces the tested character with a Boolean flag, so the character must be saved elsewhere, if it is to be used again. The flag is either the defined TRUE or the defined FALSE, which allows the proper testing of ASCII <NULL>.

Not all of the routines are limited to the classical Eaker form of CASE . It was necessary to use the additions developed by Monroe. Since these definitions do not appear in all versions of FORTH, I have included them with this article.

Notice that the Eaker part of the CASE structure has to be different, depending on whether you are using fig-FORTH or FORTH-83. If you have trouble compiling CASE, it may be because you are using the wrong version. However, the Monroe extensions to the CASE structure are the same in both versions of FORTH.

DIFFERENCES AMONG VERSIONS of FORTH

As I have said before, there are three "official" versions of FORTH floating around in the 68xx world. Fortunately, these FORTHs are more alike than they are different, so most programs written for one version will run on another version. Normally, the only changes will be obvious ones where a definition has a new name.

However, there was a major change in one type of operation. In fig-FORTH, there are so-called "state smart" words, but this was changed for FORTH-83 (I am not sure about FORTH-79)."State smart" means that FORTH knows when it is in the compiling state and when it is in the interpreting state. This is important, because a non-state smart FORTH must have different words for some operations, depending on its current state. An example is the FORTH word ' (pronounced "tick") which fetches the address of the next word in the input stream. In fig-FORTH, ' has only one form, but in FORTH-83, ' must be used while interpreting, but ['] must be used while compiling!

State smart words work in fig-FORTH, because there is a user variable called STATE , which is checked whenever a state sensitive word is encountered. If this word contains a FALSE , then one version of the definition is executed; otherwise, another version is executed.

I bring up the subject because there are some other words to look out for when changing from one

version of FORTH to the other. I'll try to provide a complete list of them at another time. But, if you have a program which worked fine in fig-FORTH, but cannot be compiled in FORTH-83, this may be the problem. You can recognize one of the symptoms as compiler failure because of an empty stack.

Two cases in point are shown in this article. Both ASCII and the CASE structure have to be changed to work with fig-FORTH or with FORTH-83. Notice how the loss of state smartness has complicated ASCII . Only one definition is sufficient for fig-FORTH, but two definitions have to be used to cover all bases in FORTH-83.

```
ASCII
```

ASCII is a somewhat ambiguous word, since it is not really necessary. It is one of those words you use for convenience, rather than any other reason. ASCII simply takes the next character in the input stream and places it on the Data Stack. You could do the same thing by simply looking up the ASCII value of the character and placing it in the input stream. In other words, ASCII Y is the same as 89 in decimal or 59 in hex. But that is precisely the point; if you don't use the word ASCII , you must know which number base you are working in, as well as the ASCII value. Furthermore, later debugging is simplified by having ASCII Y in the program text instead of a mysterious number which must be looked up in a usually-not-so-handy table.

ASCII in fig-FORTH can be used directly from the keyboard, or within a colon definition. On the other hand, in FORTH-83, 'ASCII is the word to be used from the keyboard, but ASCII is the form to be used within a colon definition. If you try to use ASCII from the keyboard in FORTH-83, you will get an error signal. Even if you remove ?COMP from the definition of ASCII, you will still get the error signal. The real problem is in the phrase [COMPILE] LITERAL , which simply bombs from the keyboard; but without it, you cannot compile ASCII .

By the way, the ' in 'ASCII in this context is "prime" and not "tick". Don't get confused by the similar appearence; I chose ' for this definition in order to make sure that the character would appear on any keyboard. Go ahead and change the ' to something else, if you prefer, just change it in both definitions.

CASE IN FORTH

A very flexible CASE structure for FORTH is available in the public domain. The version of CASE most often used is the one offered by Dr. C. E. Eaker in "FORTH DIMENSIONS", published by the FORTH Interest Group. Later on, A. J. Monroe presented an expansion to Eaker's work in the same publication. A number of other people have also offered CASE structures, but none have had the popularity of Eaker's.

A version of this CASE structure is listed here. Eaker's work was modified by W. M. Federici to work with FORTH-83. I found that Monroe's extensions worked as well for FORTH-83 as they did for fig-FORTH. Unfortunately, it was necessary to list the definitions with very little comment in order to be sure of enough space to get it all in. If you need to add CASE to your FORTH, just copy it and worry about understanding it later.

Notice that there are four essential words to the basic CASE structure, and the compilation will bomb if any are omitted or out of order.

CASE introduces the structure by making sure that it is within a colon definition by using ?COMP . The current Return Stack pointer is saved for later use during compilation. A " 4 " is put on the Data Stack for syntax checking.

OF uses " 4 ?PAIRS " to insure that it follows either CASE or ENDOF . The comparison with the index value is made with " = " and a conditional branch is set up. If the match is found to be TRUE , then DROP is used to clear the Data Stack; otherwise, the byte is left for the next comparison. A " 5 " is put on the Data Stack for syntax checking.

ENDOF uses " 5 ?PAIRS " to insure that it follows OF . It then compiles the branch to ENDCASE and puts a " 4 " on the Data Stack for syntax checking.

ENDCASE uses " 4 ?PAIRS " to insure that it follows either CASE or ENDOF . A DROP is used to clear the Data Stack and all of the conditional branches are resolved. The Return Stack pointer is restored, and the CASE structure is finished.
restored, and the CASE structure is finished.

Any expression between the last ENDOF and ENDCASE will be executed only if none of the selector keys matched. This is the proper location for the "otherwise" or default expression.

As you can see, this is as versatile a CASE structure as can be found, especially when you add Monroe's extensions for greater than, less than, and lower-upper limits. By the way, in the last definition, the lower limit must be given before the upper limit, or this section will never execute.

ADAPTING TO YOUR SYSTEM

You will have to know which type of FORTH you are using in order to know which version of the listing to enter. If you are not using Federici's "FF9", you are probably using some version of fig-FORTH. I think that most versions for the CoCo are advertised to be fig-FORTH. Test the fig-FORTH definition of ASCII from the keyboard; if it compiles, you probably have fig-FORTH. If that doesn't work, then type in the other. One should

work; if not, please let me know. Certainly, do not try to load both versions at the same time, since only the last one typed could possibly work.

Sterns Electronics FORTH for the CoCo is not a true fig-FORTH. It appears to be very much like FORTH-79; I have not had an opportunity to test it fully, but I think that it would run the fig-FORTH version shown here.

As I have said before, these listings are written as FLEX text files. If you prefer, there is no reason why they could not be entered as conventional FORTH screens. Just don't split definitions across screens, as that is very poor practice; it makes debugging more difficult than it should be.

FF9 on the CoCo

Good news! I just got word that Wilson Federici's "FF9" will now run on the CoCo with SK*DOS or FLEX. *I have tried it with DATA-COMP FLEX and FF9 ran like the proverbial charm.*

As a result of my misunderstanding of how the new typesetting program works, a couple of illustrations in the September, 1986 column got messed up. I have rewritten them here in hopes that I have finally got it right. Correction #1 shows how the coding of a definition is organized, and correction #2 shows how the FORTH stacks are organized. I am sorry for the problem, and I hope that you were still able to get something useful from the column.

EOF

Corrections:

```
   This word is compiled to:

   Machine Code              Function
      5307         Linkage address to the previous word
        82         Name letter count OR $80
      41C1         AA with last letter OR $80
      11F5         Process a "colon" definition
      0120         Duplicate the top stack number
      0ICF         Add the two top stack numbers
      0E76         Send <CR/LF> to the display device
      02C6         Send the top stack number to the display
      003C         End a "colon" definition

CORRECTION #1: The structure of a typical definition.

  EOF

    FORTH and the 6809 almost appear to be made for each other,
since FORTH makes use of four 16-bit pointers/stacks.  These
usually are:

    IP      Y reg., points to next executable word
    SP      U reg., points to top of Data Stack
    RP      S reg., points to top of Return Stack
    W       X reg., points indirectly to word being executed

The D, X, and CC registers can generally be used freely within a
definition without concern, but the other three registers must
be stored before use and recovered before exiting from a
definition.

CORRECTION #2: Pointer and stack usage.

  EOF
```

```
( ********************************************************** )
( Constants, FORTH-83                                        )
( ********************************************************** )

  -1 CONSTANT TRUE
   0 CONSTANT FALSE

( ********************************************************** )
( ********************************************************** )
( ASCII   converts a text character into a literal           )
( R. Weisling, FORTH DIMENSIONS, III/3, p. 72                )
( Adapted to FORTH-83 by M. D. Lurie                         )
(       'ASCII cannot be used within a colon definition!     )
(       ASCII must be used within a colon definition!        )
( The character must be delimited by blanks:                 )
(       : YY ASCII A EMIT ; -> A at display                  )
( ********************************************************** )

: 'ASCII ( --- number )
    BL WORD 1+ C@ ;

: ASCII             ( convert a character into a literal     )
    ?COMP 'ASCII [COMPILE] LITERAL ; IMMEDIATE

( ********************************************************** )
( ********************************************************** )
( CASE structure                                             )
( C. E. Eaker, FORTH DIMENSIONS, II/3, pp. 37-40             )
( Adapted to FORTH-83 by M. M. Federici                      )
( ********************************************************** )

: CASE  ?COMP CSP @ SP@ CSP ! 4 ; IMMEDIATE

: OF    4 ?PAIRS COMPILE OVER COMPILE = COMPILE ?BRANCH
    >MARK   COMPILE DROP  5  ; IMMEDIATE

: ENDOF  5 ?PAIRS  COMPILE BRANCH >MARK   SWAP
    >RESOLVE 4 ; IMMEDIATE

: ENDCASE  4 ?PAIRS COMPILE DROP  BEGIN SP@ CSP @ = 0=
   WHILE   >RESOLVE  REPEAT  CSP ! ; IMMEDIATE

( ********************************************************** )
( A J Monroe, FORTH DIMENSIONS, III/6, p. 167                )
( ********************************************************** )

: I<OF)
    OVER >
      IF DROP TRUE
      ELSE FALSE THEN ;

: <OF               ( LESS THAN test                         )
    4 ?PAIRS COMPILE I<OF! COMPILE ?BRANCH
    HERE 0 , 5 ; IMMEDIATE

: (>OF)
    OVER <
      IF DROP TRUE
      ELSE FALSE THEN ;

: >OF               ( GREATER THAN test                      )
    4 ?PAIRS COMPILE (>OF! COMPILE ?BRANCH
    HERE 0 , 5 ; IMMEDIATE

: RANGE
    >R OVER DUP R> I+ <
      IF SWAP 1- >
        IF DROP TRUE
        ELSE FALSE THEN
      ELSE DROP DROP FALSE THEN ;

: RNG-OF            ( inclusive RANGE test                   )
    4 ?PAIRS COMPILE RANGE COMPILE ?BRANCH
    HERE 0 , 5 ; IMMEDIATE

( ********************************************************** )
( ********************************************************** )
( CHARTEST.FTH                                    ROL  07/11/86 )
(     FORTH routines which accomplish the same job as the    )
(     corresponding C routines:                              )
(          isalpha     isalnum     isascii     iscntrl       )
(          isdigit     islower     isupper     isprint       )
(          ispunct     isspace                               )
( ********************************************************** )
```

```
( ?ISALPHA        test for A-Z or a-z            RDL  01/10/86 )
( ********************************************************** )

: ?ISALPHA          ( c --- bool   )
    CASE
      ASCII A ASCII Z RNG-OF TRUE ENDOF
      ASCII a ASCII z RNG-OF TRUE ENDOF
    ENDCASE
      DUP TRUE = NOT IF FALSE THEN ;

( ********************************************************** )
( ?ISALPHANUMERIC test for 0-9 or A-Z or a-z      RDL  06/04/86 )
( ********************************************************** )

: ?ISALPHANUMERIC ( c --- bool   )
    CASE
      ASCII 0 ASCII 9 RNG-OF TRUE ENDOF
      ASCII A ASCII Z RNG-OF TRUE ENDOF
      ASCII a ASCII z RNG-OF TRUE ENDOF
    ENDCASE
      DUP TRUE = NOT IF FALSE THEN ;

( ********************************************************** )
( ?ISASCII        test for an ASCII character    RDL  01/10/86 )
( ********************************************************** )

: ?ISASCII          ( c --- bool   )
    128 < ;

( ********************************************************** )
( ?ISCNTRL        test for a control character   RDL  01/10/86 )
( ********************************************************** )

: ?ISCNTRL          ( c --- bool   )
    CASE
      32 <OF TRUE ENDOF
      127 OF TRUE ENDOF
    ENDCASE
      DUP TRUE = NOT IF FALSE THEN ;

( ********************************************************** )
( ?ISDIGIT        test for 0-9                   RDL  01/10/86 )
( ********************************************************** )

: ?ISDIGIT          ( c --- bool   )
    CASE
      ASCII 0 ASCII 9 RNG-OF TRUE ENDOF
    ENDCASE
      DUP TRUE = NOT IF FALSE THEN ;

( ********************************************************** )
( ?ISLOWER        test for a-z                   RDL  01/10/86 )
( ********************************************************** )

: ?ISLOWER          ( c --- bool   )
    CASE
      ASCII a ASCII z RNG-OF TRUE ENDOF
    ENDCASE
      DUP TRUE = NOT IF FALSE THEN ;

( ********************************************************** )
( ?ISUPPER        test for A-Z                   RDL  01/10/86 )
( ********************************************************** )

: ?ISUPPER          ( c --- bool   )
    CASE
      ASCII A ASCII Z RNG-OF TRUE ENDOF
    ENDCASE
      DUP TRUE = NOT IF FALSE THEN ;

( ********************************************************** )
( ?ISPRINT        test for <SP>--                RDL  01/10/86 )
( ********************************************************** )

: ?ISPRINT          ( c --- bool   )
    CASE
      BL ASCII ~ RNG-OF TRUE ENDOF
    ENDCASE
      DUP TRUE = NOT IF FALSE THEN ;

( ********************************************************** )
( ?ISPUNCT        test for punctuation           RDL  06/04/86 )
( ********************************************************** )

: ?ISPUNCT          ( c --- bool   )
    CASE
      ASCII ! ASCII / RNG-OF TRUE ENDOF
      ASCII : ASCII @ RNG-OF TRUE ENDOF
      ASCII [ ASCII ` RNG-OF TRUE ENDOF
      ASCII { ASCII ~ RNG-OF TRUE ENDOF
    ENDCASE
      DUP TRUE = NOT IF FALSE THEN ;

( ********************************************************** )
( ?ISSPACE        test for <SP><TAB><LF><FF><CR> RDL  01/10/86 )
( ********************************************************** )

: ?ISSPACE          ( c --- bool   )



          CASE
            32 OF TRUE ENDOF                ( <SP>          )
            09 OF TRUE ENDOF                ( <TAB>         )
            10 OF TRUE ENDOF                ( <LF>          )
            12 OF TRUE ENDOF                ( <FF>          )
            13 OF TRUE ENDOF                ( <CR>          )
          ENDCASE
            DUP TRUE = NOT IF FALSE THEN ;


EOF


( ********************************************************** )
( Constants, FIG-FORTH                                     )
( ********************************************************** )

   1 CONSTANT TRUE
   0 CONSTANT FALSE

( ********************************************************** )
( ********************************************************** )
( ASCII   converts a text character into a literal         )
( R. Weisling, FORTH DIMENSIONS. III/3, p. 72              )
( The character must be delimited by blanks:               )
(       : YY ASCII A EMIT ; -> A at display                )
( ********************************************************** )

: ASCII   ( --- number )
    BL WORD HERE 1+ C@
    [COMPILE] LITERAL ; IMMEDIATE

( ********************************************************** )
( ********************************************************** )
( CASE structure                                           )
( C. E. Eaker, FORTH DIMENSIONS, II/3, pp. 37-40           )
( Modified by A J Monroe, FORTH DIMENSIONS, III/5, p. 107  )
( Modified by R. D. Lurie                                  )
( ********************************************************** )

: CASE  ?COMP CSP @ !CSP 4 ; IMMEDIATE

: (OF)  OVER = IF DROP TRUE ELSE FALSE THEN ;

: OF    4 ?PAIRS COMPILE (OF) COMPILE 0BRANCH
    HERE 0 , 5 ; IMMEDIATE

: ENDOF  5 ?PAIRS  COMPILE BRANCH HERE 0 ,   SWAP
    2 [COMPILE] THEN 4 ; IMMEDIATE

: ENDCASE  4 ?PAIRS COMPILE DROP  BEGIN SP@ CSP @ - 0 =
   WHILE 2 [COMPILE] THEN  REPEAT  CSP ! ; IMMEDIATE


EOF
```

A CUSTOM FLEX

Karl Lunt
7349 W. Canterbury
Peoria, AZ 85345

The accompanying article describes the steps I used in generating a new version of FLEX for a recently-completed 6809 system. The hardware is a 68B09 with 64K, DSDD 2/3 height Remex drives, two serial ports and counter/timer. It is a wirewrapped board that fits onto a 5.25" disc drive, using PLAs to reduce the chip count.

The new FLEX system for this machine was put together over several weekends, using an older 6809 system as the developement (host) system. This software development taught me a great deal about the FLEX operating system. I have even more respect for the job TSC did with FLEX than I did before I started this project

FLEX is a versatile and elegant solution to the problem of an operating system for a single user on a single machine. The fact that I was able to customize a FLEX for one machine into a FLEX for a second, very different (with regards to I/O addressing) machine in so little time, speaks highly for the care that TSC put into their original design of FLEX.

With the prices of 68XX chips and hardware so low, and considering how easily FLEX can be customized for new hardware, a do-it-yourself 64K (or more) 6809 system can be done by nearly any experienced hobbyist. For me, this is a large measure of the enjoyment I get from working with the 6809.

Another large part of my fun with the 6809 systems concerns the excellent job done by you, your staff and your readers. I have read (and subscribed) to many computer magazines in the past. 68' Micro Journal is my only remaining subscription and it is the only magazine that still carries for me the enthusiasm and excitement of the early days of computing. In particular, the input from readers around the world makes 68' Micro Journal a very unique magazine.

Speaking of which, the thing that got me started on this article was a letter/article appearing in the October 1986 issue from Neil Preston of Zimbabwe(!). Neil talked about his efforts in putting together a 6809 system and expressed

First of all, I wish Neil the best of luck in putting his system together. He sounds like he is making good progress. He talks about changing from 8" to 5.25" drives. I don't know much about the inner workings of the Uniboard, but if it uses the 2793 style FDC, he should be able to alter the disc data clock by switching pin 25 (ENMF*) high or low. Details can be found in the Western Digital Storage Management Products Handbook or a WD2793 data sheet. Note that if his system uses the 2795/2797 FDC instead, this technique is not immediately available, as that pin is used for side select output (SSO) instead.

About his concern that he will be left behind by the 68XXX technology; that will probably not be a problem for a while. A 6809 system still does everything I need to do for my hobby, even though I have an Apple II sitting on the next desk and I use a 68020 graphics system at work

*Editor's Note: Thanks Karl for the nice letter and your article on converting FLEX. I guess that we will certainly be around as long as there are folks who, like yourself, are willing to share and help those less knowledgeable. FLEX and SK*DOS are still the finest single tasking, single user systems around. FLEX may have a few 'gotchas' here and there, but over the years, somewhere in 68 Micro Journal, I believe someone addressed those subjects. If not, they will, just wait.*

Thanks Karl, I and thousands of readers certainly do appreciate all of you who contribute.

68 Micro Journal isn't just all paper and ink, It is all those who contribute, thats really what 68 Micro Journal is.

DMW

The task of customizing a FLEX operating system for a 6809 computer is not trivial, but is within the reach of an assembly language programmer with the proper tools. These tools include a host 6809 system on which to develop the new FLEX system, as well as an assembler, debugger and editor. Additionally, the programmer should have a copy of the FLEX Adaptation Guide, written by Technical Systems Consultants. Attempting to create a new FLEX system without this guide is asking for a lot of headaches.

The system creation consists of three major tasks. The FLEX system itself must be customized for the target machine. Secondly, the NEWDISK formatter program must be modified to properly interface with the system's disc drives. Finally, an appropriate one-sector loader must be prepared and added to the new NEWDISK program so the discs formatted on the new system may be booted by that system.

The FLEX system

The FLEX operating system is divided into three major components. FLEX.COR is the main body of the FLEX system and handles all file control blocks (FCBs) used by the system. It also processes user input, generates output and controls disc accesses at the highest level.

FLEX' interaction with the user's terminal is handled by the routines in the FLEXIO.BIN file. Similarly, FLEX' interaction with the system discs is done with the FLEXDISC.BIN routines. These last two .BIN files must be customized for the target system, as the locations of status and command registers vary among designs.

FLEX.COR is normally supplied with every FLEX system. If you are using FLEX on the host machine, you should already have a copy of FLEX.COR available. The remainder of the FLEX system is built by APPENDing the appropriate FLEXIO and FLEXDISC binary files onto the FLEX.COR file. The file created by this APPEND operation is named FLEX.SYS and is the new FLEX system.

My target system has DSDD 5.25" 40-track drives. The floppy disc controller (FDC) is a WD2793. The FDC status/command registers occupy memory from $E018-$E01C, while the drive, side and density select is done through address $E014. This is quite different from the setup in my host system, so I had to create a new FLEXDISC.BIN file.

As I did not get the source for my host system's FLEXDISC.BIN, I used a disassembler to generate an assembler source file of the code in locations $DE00 to $DFFF, the area in a standard FLEX system reserved for the disc primitive routines. If you must start completely from scratch in developing your FLEXDISC source file, you can use the Adaptation Guide to help your development. Appendix G of the Guide gives a listing of the disc driver routines for the SWTPC MF-68 disc system and makes an excellent starting point.

After I disassembled my host's disc drivers, I changed the proper routines to reflect the hardware described above. In particular, I modified the READ and WRITE routines to change density if a "record not found" error is encountered. This causes FLEX to try different densities automatically if it can't find the proper sector. This means that my new FLEX system handles double/single density discs without instructions from the user. Details on this technique are available in the Guide on pages 40-41.

An improved SEEK routine was also needed. The new SEEK uses the proper bits in the system drive select address ($E014) to control side and density select. The selection is made based on the desired sector number and the status of the disc driver's internal density table. Again, consult the Guide, page 41 for details.

Incidentally, the SEEK routine in the SWTPC drivers contains a $1B as the seek command issued to the controller (a 1771 FDC). This seek command causes a step rate of 20ms on a 1Mhz system and occurs in the area of $DE80. Changing this value to $1A will produce a 10ms step rate and $19 will produce a 6ms step rate. On my host system, I patch this code in FLEX at boot-time by overwriting it with a 1-byte long program ORGed at the appropriate address. On the new system, I simply wrote the source code to use the step rate I wanted.

The final source code for the disc drivers was assembled and the object written to my new FLEXDISC.BIN file. The terminal primitive file FLEXIO.BIN was created by SAVEing the desired area of memory ($D370-$D3FF) from my host system to disc. This was possible because the terminal I/O sections for both machines are identical. If you need to create customized terminal drivers for your new FLEX system, consult the Guide, pages 6-8.

The three files that make up the new FLEX system are APPENDed together into a larger single file with a command similar to the following:

```
APPEND FLEX.COR FLEXDISC.BIN
FLEXIO.BIN MIKEFLEX.SYS
```

A word about transfer addresses: the FLEX system is loaded from disc as any other binary file, with one important exception. It must have only one transfer address and this transfer address must be the last data stored in the file. As described above, FLEX.COR has no transfer address in it, FLEXDISC.BIN was assembled without an address label on the final END line in the source file, and FLEXIO.BIN was written to disc with a command like:

```
SAVE FLEXIO.BIN D370 D3FF CD00
```

Thus, when all three files are appended together, the final transfer address is $CD00. This should be verified with a command such as MAP, that looks at the addresses affected by .BIN or .CMD file.

The NEWDISK command

The next step in creating the new FLEX system is to modify the NEWDISK (or similar) command that formats a floppy disc. In my case, I had two things to worry about. First, the method used for selecting density in my new system did not match the method used in the SWTPC-based NEWDISK program that came with my original FLEX package. Second, I needed to change the bootstrap loader that is written to track 0, sector 1 by the formatter routine as one of its final steps in formatting a disc.

The first task was done quickly, using a disassembly of the original NEWDISK command to locate the few locations of interest. Then, the DEBUG program was used to GET the NEWDISK.CMD file into memory, the necessary areas were patched, and the altered code was SAVEd back to disc, this time as FORMAT.BIN. The second of these tasks took a little more work and the Guide was very valuable for this phase.

When a FLEX disc formatter finishes formatting a disc, it also writes a small (one or two sector) bootstrap loader to the disc, starting at track 0, sector 1. Later, when the system is booted from this disc, the ROM-based bootstrap loader loads in the disc-based bootstrap and transfers to it.

This disc-based bootstrap is really little more than SEEK and READ routines, with a few file loader instructions in between. However, the READ and SEEK routines must select the side and density of the floppy disc in exactly the same way used by FLEX. This means that the new FLEX system will be bootable from either a single-density or double-density disc, with density and side selections done automatically.

Moreover, the disc-based bootstrap must be memory-resident when the formatter needs it. This means that both programs must load into memory from one command.

The disc-based bootstrap routine was written to reside in memory from $C100-$C1E8 and contained no transfer address. I was able to write the bootstrap such that it fitted into only one sector. If you need more than one sector to hold your disc-based bootstrap, consult the Guide for details on how to handle the extra sectors. More information can be found in a disassembly of the SWTPC NEWDISK program, if you have it available.

All that remained in finishing the FORMAT command was to use the DEBUG program to patch in the address of the new disc-based bootstrap module (ORGed at $C100), save the FORMAT file back to disc, then APPEND the LOADER.BIN file to it with a command such as:

```
APPEND FORMAT.BIN LOADER.BIN
FORMAT.CMD
```

Since my target system could boot a single-side, single-density disc written by my host system, I now used the target system to format a double-sided disc, then used the EXAMINE command (a single sector read/write program) to verify that the proper bootstrap had been written to track 0, sector 1. This completed the development of the new FLEX system, but I still had one last thing to do.

The ROM bootstrap

This last step involved altering the ROM-based bootstrap for the target machine. The changes were needed because the old ROM bootstrap had a conflict between where it located the stack and where it placed the disc-based bootstrap as it read it in. Though this was a minor change, I also took the time to make a fairly significant change.

The ROM-based bootstrap described by TSC in their FLEX User's Manual (page 3.6) has a very annoying flaw. When this code is executed, it does a brute-force read of 256 bytes from the disc and jumps to $C000, without verifying that the data was read properly. It doesn't even check to see if it read 256 bytes! This means that sometimes, depending on conditions such as disc speed, sunspot activity, recent stock market prices, etc., my system simply will not boot.

Besides rewritting this bootstrap to jump to $C100 (the location of my disc-based bootstrap following a boot), I also added code to verify that no error status codes were returned by the FDC and that a full 256 bytes were read. If either of these tests fail, I issue a restore and try again.

After the new bootstrap was burned into EPROM and installed in the target system, I was ready to finally put all the pieces together. A single-sided disc containing the new FLEX system (called MIKEFLEX.SYS) and the FORMAT command was generated on the host system. The LINK command was used to connect MIKEFLEX.SYS to the disc-based bootstrap loader and the disc was ready for booting on the target system.

Pressing reset on the target system and executing the bootstrap code loaded the new FLEX system into memory and it executed properly. I felt there should have been more fanfare to this first booting of a new system than the simple '+++' FLEX prompt, but it WAS exciting to see the boot take place!

The last thing to test was FORMATting a disc on my new system and attempting to boot it. This worked exactly as expected and my new FLEX system is up and running. In fact, I wrote this article on it.

Special thanks to Brian Bierer of Phoenix, AZ, for his help and suggestions on the disc primitives. He helped make the job go easier.

EOF

MICROWARE EXPANDS....AGAIN

On June 13, 1986, Microware Corp, developers for OS-9, celebrated ground breaking ceremonies for the second major expansion, in less than three years.

Ken Kaplan, President and CEO reported that due to the pressures of additional personnel and greatly increasing customer acceptance for OS-9, and other Microware software products, the building and grounds expansion was both a current necessity and a commitment to the future. The new facility will encompass over 8,000 square feet and will house the expanding administrative and marketing departments. They expect to be completely moved into the new facilities by January 1987.

Recent additions to the Microware staff are:

Ken Mizuno, software engineer. Ken is originally from the Microware Japan Branch. Ken will be working on NFM and CD-I software tools.

Richard Russell, software engineer, *came up through the ranks*, so to speak, and still uses his home system, running Microware C, in designing compilers and other software.

Kim Gegner, receptionist, is a native of Des Moines and formerly worked in the research and data entry department of a Des Moines bank.

Paula Henderson, production clerk, is still attending the University of Northern Iowa. Paula works in the production department.

Christy Longstaff, cook, is one of the more appreciated members of the staff, I imagine. The personnel at Microware have their own kitchen and dining area. Christy is also a Des Moines native.

Robert Wolf, electronics engineer, with a background in digital design and machine language programming. Robert has already developed an application package for cooperative buying clubs.

We wish then all success and happiness in their new vocations. And, it is nice to see Microware expanding at such a rapid pace. Guess they must be doing something right. Congratulations all!

DMW

Ken Kaplan, President and CEO of Microware Corp breaks ground with the first shovel of dirt. Microware staff members look on.

The new 8,000 square foot expansion to the Microware complex. Situated in a wooded and quite area of Des Moines, the Microware facilities represent the modern expression of 'high tech' urban location.

DISASSEMBLERS

SUPER SLEUTH from Computer Systems Consultants Interactive Disassembler; extremely *POWERFUL!* Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems.

Color Computer SS-50 Bus (all w/ A.L. Source)
CCD (32K Req'd) Obj. Only $49.00
F, $99.00 - CCF, Obj. Only $50.00 U, $100.00
CCF, w/Source $99.00 O, $101.00
CCO, Obj. Only $50.00

DYNAMITE+ -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00 - CO, Obj. Only $ 59.95
F, " " $100.00 - O, object only $150.00
U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from Windrush Micro Systems -- By Graham Trott. A combination Editor Compiler Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide.

F, CCF - $198.00

PASC from S.E. Media - A Flex9 Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package come complete with source (written in PASC) and documentation.

FLEX $95.00

WHIMSICAL from S.E. MEDIA Now supports *Real Numbers*. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. *Normally produces 10% less code than PL/9.*

F and CCF - $195.00

KANSAS CITY BASIC from S.E. Media - *Basic for Color Computer OS-9* with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFT$, RIGHT$, MID$, STRING$, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.

CoCo OS-9 $39.95

C Compiler from Windrush Micro Systems by James McCosh. Full C for FLEX except bit-fields, including an Assembler. *Requires the TSC Relocating Assembler if user desires to implement his own Libraries.*

F and CCF - $295.00

C Compiler from Introl -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.

FLEX, CCF, OS-9 (Level II ONLY), U - $575.00

PASCAL Compiler from Lucidata -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.

F and CCF 5" - $99.95 F 8" - $99.95

PASCAL Compiler from OmegaSoft (now Certified Software) -- For the *PROFESSIONAL*; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.

F and CCF - $425.00 - One Year Maint. $100.00
OS-9 68000 Version - $900.00

K BASIC - from S.E. MEDIA -- A "Native Code" BASIC Compiler which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.

FLEX, CCF, OS-9 Compiler /Assembler $199.00

CRUNCH COBOL from S.E. MEDIA -- Supports large subset of ANSII Level 1 *COBOL* with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System. *A very popular product.*

FLEX, CCF; Normally $199.00
Special Introductory Price $99.95

FORTH from Stearns Electronics -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!

Color Computer ONLY - $58.95

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends–

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

* OS-9 is a Trademark of Microware and Motorola
* FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from S.E. Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $59.95, U - $89.95

LOW COST PROGRAM KITS from Southeast Media -- The following kits are available for FLEX on either 5 or 8 inch disk.

1. **BASIC TOOL-CHEST $29.95**
   BLISTER.CMD: pretty printer
   UNEXREF.BAS: line cross-referencer
   REMPAC.BAS, SPCPAC.BAS, COMPAC.BAS: remove superfluous code
   STRIP.BAS: superfluous line-numbers stripper
2. **FLEX UTILITIES KIT $39.95**
   CATS.CMD: alphabetically-sorted directory listing
   CATD.CMD: date-sorted directory listing
   COPYSORT.CMD: file copy, alphabetically
   COPYDATE.CMD: file copy, by date-order
   FILEDATE.CMD: change file creation date
   INFO.CMD (& INFOGMX.CMD): tells disk attributes & contents
   RELINK.CMD (& RELINK82): re-orders fragmented free chain
   RESQ.CMD: undeletes (recovers) a deleted file
   SECTORS.CMD: show sector order in free chain
   XL.CMD: super text lister
3. **ASSEMBLERS/DISASSEMBLERS UTILITIES $39.95**
   LINEFEED.CMD: 'modularise' disassembler output
   MATH.CMD: decimal, hex, binary, octal conversions & tables
   SKIP.CMD: column stripper
4. **WORD - PROCESSOR SUPPORT UTILITIES $49.95**
   FULLSTOP.CMD: checks for capitalization where required
   BSTYCIT.BAS (.BAC): Stylo to dot-matrix printer program
   NECPRINT.CMD: Stylo to dot-matrix printer filter code
5. **UTILITIES FOR INDEXING $49.95**
   MENU.BAS: selects required program from list below
   INDEX.BAC: word index
   PHRASES.BAC: phrase index
   CONTENT.BAC: table of contents
   INDXSORT.BAC: fast alphabetic sort routine
   FORMATER.BAC: produces a 2-column formatted index
   APPEND.BAC: append any number of files
   CHAR.BIN: line reader

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F and CCF, U - $25.00, w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 Regular $149.95
SPECIAL INTRODUCTION OFFER *$69.95*

DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.

Level I OS-9 obj, $79.95; w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX Format so it can be used normally by FLEX; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk

O - 6809/68000 $79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and errors messages.

OS-9 $85.00

HIER from S.E. Media - *HIER is a modern hierarchal storage system for users under FLEX.* It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. Using HIER a regular (any) FLEX disk (8 - 5 - hard disk) can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modern day

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends--

F - FLEX, CCF - Color Computer FLEX
O - OS-9, CCO - Color Computer OS-9
U - UniFLEX
CCD - Color Computer Disk
CCT - Color Computer Tape

*OS-9 is a Trademark of Microware and Motorola
*FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA

solution that all current large systems use. **Each directory looks to FLEX like a regular file, except they have the extension '.DIR'.** A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special install package is included to install HIER to your particular version of FLEX. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!

** Introduction Special * $69.95*

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller disks. FLEX utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORB.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

NEW

COPYCAT from Lucidata -- *Pascal NOT required.* Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.

F and CCF 5" - $50.00 F 8" - $65.00

VIRTUAL TERMINAL from S.E. Media - Allows one terminal to do the work of several. The user may start as many as eight task on one terminal, under *VIRTUAL TERMINAL* and switch back and forth between task at will. No need to exit each one; just jump back and forth. Complete with configuration program. The best way to keep up with those background programs.

O & CCO - obj. only - $49.95

FLEX DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs.

ALL Utilities include Source (either BASIC or A.L. Source Code).

F and CCF - $50.00
BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

FLEX, CCF, OS-9, UniFLEX; with complete Source $100.00 without Source $50.00
UniFLEX 68000 with complete Source $100.00

X-TALK from S.E. Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transferred between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.

X-TALK Complete (cable, 2 disks) $99.95
X-TALK Software (2 disks only) $69.95
X-TALK with CMODEM Source $149.95

XDATA from S.E. Media - A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.

U - $299.99

ASSEMBLERS

ASTRUK09 from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.

F, CCF - $99.95

Macro Assembler for TSC -- The FLEX STANDARD Assembler.

Special -- CCF $35.00; F $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. GeneOrate OS-9 Memory modules under FLEX.

FLEX, CCF, OS-9 $99.00

Relocating Assembler/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.

F, CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.

F, CCF - $75.00
XMACB -- MACE w/Cross Assembler for 6800/1/2/3/8 *F, CCF - $98.00*

Availability Legends--

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

* OS-9 is a Trademark of Microware and Motorola
* FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

EDITORS & WORD PROCESSING

JUST from S.E. Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:

Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source:FLEX - CC

Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .ep .ce etc.) Great for your older text files. The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p ,u ,y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX only - F & CCF - $49.95
Disk Set (2) - F & CCF & OS9 (C version) - $69.95
OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX $129.50
** SPECIAL INTRODUCTION OFFER * $79.95*
SPECIAL PAT/JUST COMBO (w/source) FLEX $99.95
OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

Regular $129.95
SPECIAL INTRODUCTION OFFER FLEX $69.95

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entires lines, etc. Complete with over 25 different CRT terminal configuration overlays.

FLEX, CCF, STAR-DOS Regular $69.95
Limited Special Offer: $39.95

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

SPELLB "Computer Dictionary" from S.E. Media -- OVER 150,000 words! *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - $99.95,
F or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES 6809 CCF and CCO - $69.95,
F or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES6809 CCF and CCO - $59.95,
F or O - $79.95, U - $129.95

STYLO-PAK -- Graph + Spell + Merge Package Deal!!!

F or O - $329.95, U - $549.95
O, 68000 $595.00

GAMES

RAPIER - 6809 Chess Program from S.E. Media -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. **This is one of the strongest CHESS programs running on any microcomputer,** *estimated USCF Rating 1600+ (better than most 'club' players at higher levels)*

F and CCF - $79.95

!!! Please Specify Your Operating System & Disk Size !!!

Availability Legends–

F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

* OS-9 is a Trademark of Microware and Motorola
* FLEX is a Trademark of Technical Systems Consultants

** Shipping **

Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA

68 MICRO JOURNAL Writers Guide

Please note the following guidelines for articles and other disk submitted material for publication in 68 Micro Journal. Due to our new system of entering and editing text for 68 Micro Journal, the following brief outline and example is made available for your information, should you decide to submit material for publication.

We accept material on diskettes in FLEX, OS-9, RS, WORD and MacWrite format, also files may be modem downloaded at 1200 baud. We need advance notice for modem delivered files. *All text files (article) should have no space-bar formatting...Please!* Spaces are of different widths in typesetting. This is due to the nature of *scaled fonts.* That is, each typeface and font size has a different width, hence, the space is different in various parts of a document. So, if you use a space to indent, or to align columns, it will be a mess when we have the typesetter output your work of love! **PLEASE...let us put in the formatting.** If a column(s) needs to line up, note that in some other manner, but ***PLEASE DO NOT*** use spaces to make your text look straight, aligned, centered, columinized, or any other text style or format.

We use a tab system that is very precise. Tabs we insert to format, align on all available boundries, as well as decimal points, etc. We need to be the ones to put the tabs in, according to how wide, type size, type face, font style, etc., O.K.?

The following example text file is as we would like to receive ALL disk files for publication.

PLEASE NOTE: Because we use either a fixed space typeface or a printer output for code source listing, complete source listing can be space formatted. However...very important...if you put fragments of source code inline with your regular text file, ***PLEASE LEAVE OUT THE SPACE FORMATTING.*** We will attempt to restructure (format) it with our tabbing system. Otherwise put sources, listings. etc. in a separate file.

Example listing:

```
This is an example of text submitted for publication in 68 Micro Journal. It is be
st that it have no carriage returns, except at the end of a paragraph. Only one (1) space between

*A* INSERT FIG. 1.0 HERE (NOW WE KNOW WHAT TO DO WITH SEPARATE LISTINGS)

sentences and no extra carriage returns (no extra line(s) between paragraphs). Extra lines sho
uld only be between subject headings as below.

THIS IS A SUBJECT HEADING
Note that there was an extra carriage return (with linefeed done automatically by most syst
ems) directly above subject heading. This is the way that your text (disk) file should look if we list it out to t
he screen.

THIS IS A SAMPLE OF INLINE CODE

!-LDA #34, X+100
!-STA TEMP
etc.
```

Please note the source inline code (inline with your text file) as opposed to a source code file that is separate from the text file. That file (separate file) can have spaces for formatting as it will be reproduced on a printer that is space fixed for the entire type style and font. The '!-' in the above source inline code is prefixed with a symbol for a space. This alerts our operator to attempt a reasonable tab format on the code, or whatever. It might be any sort of column dependant text. However, WE must be the ones to format all inline text (code or otherwise).

Please note that if there is very much inline code it should have the "*INSERT HERE*" typed in (or some other indicator to let us know what is desired by you...see *A* above). Otherwise it should be a separate file to be appended to the text portion of the file.

(Clip & save for future reference)

Simple Winchester

by: Samuel I. Green Ph.D.
13052 Ferntrails Lane
Creve Coeur MO 63141

This article describes the inexpensive installation of a winchester hard disk drive into a SWTP 6809 computer. *With little change this applies to many other brands of S50 Bus systems also - DMW.* This implementation assumes that the motherboard decodes the I/O slots to have 16 addresses per slot and that the I/O range is decoded so that a 2k RAM can be addressed at $E800. Modifications to motherboard decoding are described in the Appendix.

My implementation costs less than $300 and includes a 5 megabyte drive which was available surplus for less than $100. The controller is a WD1000-05 which sells for $160 or $185 from my Western Digital distributor depending on whether you work for a company which rates a discount. A simple host interface card installs in the SS-30 bus and connects to the controller via a 40 conductor ribbon cable. The controller connects to the printed circuit board on the drive with two ribbon cables. A 34 conductor ribbon cable is daisy chained to all drives with a termination resistor DIP in the last drive just like floppy drives. The drive also has an address decode switch like a floppy to set the drive number. A 20 conductor ribbon cable handles the high speed read/write data between the controller and the winchester drive. The controller has four 20 pin connectors, and a separate 20 conductor ribbon cable is used for each drive.

Implementation costs less than $300 and includes a 5 megabyte drive

The WD1000-05 looks like a parallel port which requires eight addresses. Other winchester controllers use a SASI or SCSI interface which has handshaking lines. The WD1000-05 is simpler to hook up and simpler to get running.

Two sets of software drivers are provided to operate the winchester drive. They are compatible, and either may be used. The first called WINDRIVR is based on the article by David Graves which appeared in the October and November 1982 issues of the '68' Micro Journal, for which I remain very

grateful. His drivers were a modification to those written earlier by R. Zeff and were written for an earlier version of the Western Digital winchester controller board. I revised the code to work properly with the WD1000-05 aided by my friend Jim Babb. Other changes are to use true data rather than inverted data since my host interface board is installed in the SS-30 bus rather than the SS-50 bus. Memory consuming in-line code to read and write sectors is replaced with looping code with no noticable loss of speed.

The second set of winchester drivers called VDW is incorporated into the Scudiere drive selector which allows up to four floppy drives, four winchester drives, and one virtual drive (RAM Disk) to be used in the same system in any combination of four at one time. This is based on the excellent article by Matt Scudiere in the December 1982 issue of the '68' Micro Journal. My friend Robert Arnzen helped make the virtual disk work and added the feature which automatically finds and uses all extended addressed memory which is uniquely addressed.

Both WINDRIVR and VDW still incorporate the Graves/Zeff limitation of using each head of a winchester as a different logical drive. This sounds primitive, but it works very well in a system with only one winchester drive which has two heads. When my system boots, drive 0 is a floppy, drive 1 is virtual, drive 2 is the system winchester drive (head 0), and drive 3 is the working winchester drive (head 1). Also, flex only accomodates track numbers from 0 to 255, so higher number tracks are presently wasted in my system. It is a straightforward exercise to translate physical heads and tracks and sectors into logical track and sector numbers which flex can accomodate. The drivers provided will work with the first two heads and the first 256 tracks of whatever drive you try. Note that a very old drive like the Seagate ST-501 does not have buffered seek so that a delay must be added between track steps. The Tandon TM-501 and the Shugart SA-604 work fine.

I have recently received new winchester drivers from Leo Taylor which he offered in the June 1986 issue of the '68' Micro Journal. His drivers allow use of track numbers higher than 255 and partitioning of disks into as many as 26 volumes with any volume assignable as a Flex logical drive number.

The format command is WINFMT. It too was written by R. Zeff and modified by D. Graves. I further modified it to work with the WD1000-05 controller and true data and replaced the memory consuming in-line read and write code with looping code. The format command still has the limitation of not removing bad sectors, however I found that bad sectors were not repeatable. Often a whole track would register bad when sectors were being written out during the linking process because the track wasn't successfully formatted. Therefore I further modified the format command to reformat bad tracks until they passed without errors. A drive with a true hard error will likely hang the format routine at this point. If this happens, delete the reformat routine.

The command called PARK is used to move the head to a track number higher than the maximum useable track. It is wise to park the head before removing motor power so that the head doesn't get to come down off of its cushion of air to rest someplace important like the directory. It is mandatory that you park the drive before moving or shipping it.

The SS-30 host interface card simplifies to the circuit of Figure 1 which has only two gate packages and an octal bidirectional bus buffer. The Graves/Zeff circuit for the SS-50 bus was much more complicated. It included I/O address decoding which is handled by the motherboard, RAM at $E800 for the drivers which is a 2k x 8 static RAM in one of the ROM sockets on the SWTP MP-09A CPU board, and a memory slowdown not needed by the newer WD1000-05 controller. Figure 2 is a suggested wiring layout for the SS-30 host interface card.

The following hardware assumptions are made. The WD1000-05 controller requires eight addresses. This means that if you only have four addresses per I/O slot, your motherboard decoding must be changed. You knew it was time to do this anyway, and now you have a good reason to do it. I will outline the procedure in the Appendix. Also, 2k of RAM is assumed at $E800 for the drivers. This requires that the I/O address range must be more fully decoded. I will deal with this in the Appendix also. For the Scudiere virtual disk, all system memory must be extended addressed and fully decoded so that one memory board doesn't appear to be at more than one address. The baud rate lines of the SS-50 bus become address select lines S0 through S3 and must be separated from the baud rate lines of the SS-30 bus. The baud rate generator is then moved to the SS-30 bus. I will describe these changes in the Appendix. In my system, the baud rate generator is on the same card as the winchester host interface.

The WD1000-05 winchester controller board has the same format as the five and one-quarter inch winchester drive and has mounting holes which allow it to be installed with standoffs onto the winchester drive. In this position, the connectors of the drive and the controller match up so that they connect easily with short ribbon cables. This arrangement is shown in Figure 3. The controller board uses dual rows of pins on 0.1 inch centers for interconnection while the drive uses card edge connectors. Mating ribbon cables are easily made using readily available connectors.

The winchester drive uses the same four pin power connector with the same voltages that is used on floppy drives. The WD1000-05 controller also uses this same power connector, but requires only +5 volts. A hefty power supply is recommended, especially for the older drives which draw a lot of motor current from the +12 volt supply. An inexpensive choice is the switching power supply advertised widely by B.G. Micro. (*See DATA-COMP advertising - this issue - for excellent low cost power supplies for this project)* I have not installed my drive/controller/power supply into a nice case. That is partly how I kept the cost below $300.

The performance improvement of the winchester over floppy disk drives is amazing. In fact, the winchester takes only half again as long as the virtual disk to perform a comparable task.

APPENDI
Motherboard Modifications

This section will describe the following modifications.

1. 16 addresses per I/O slot
2. Fully decode I/O address range
3. Cut baud rate lines
4. SS-30 baud rate generator board

16 addresses per I/O slot

The early SWTP motherboards are readily converted from four addresses per I/O slot to sixteen addresses per I/O slot by performing the following changes shown in Figure 4. Connect address lines A2 and A3 from the 50 pin bus to the user defined lines UD3 and UD4 on the 30 pin bus. Be sure to remove any jumper to UD3 which was required by the floppy disk controller for drive selection.

Pins 1,2, and 3 of IC3 on the motherboard originally go to address lines A2, A3, and A4. Cut these three lines and wire them to A4, A5, and A6 respectively.

After converting to 16 addresses per slot, your system serial port is moved from slot 1 to slot 0. No other change is needed in hardware decoding or software driver since the port address of $E004 has been moved to slot zero and the incomplete decoding of the original 4 address serial board will cause the board to be addressed at $E000, $E004, $E008, as well as $E00C.

The floppy disk controller board is moved to slot 1 so that it can be addressed from $E014 to $E01B as previously. My floppy controller is jumper selectable for 4 or 16 addresses. Additional decoding will be needed on an older floppy controller board if it is not compatible with 16 addresses per slot. An example which shows the modification for the DC-2 floppy controller is shown in Figure 5.

I/O hardware plugged into other SS-30 slots is now found at new addresses, so that software drivers for this hardware must be rewritten. For example, a printer driver which accessed the A side of a PIA at $E01C,D to run a parallel printer must be reassembled to look for the PIA at $E07C,D if the interface is to remain in slot 7. Other drivers which would require address changes are modem programs or serial printer drivers which refer to an additional serial port, EPROM programer drivers, and calender/clock board drivers. Some software is adjusted by simply using SBOX. Most driver source listings have a single port address equate allowing simple change and reassembly.

Fully decode I/O addressing range

Older SWTP motherboards have incomplete I/O decoding which allows the I/O addressing range to extend up into the $E800 area where we want to put a RAM on the CPU board. The simplest way to use an $E800 RAM is to connect A11 from the 50 pin bus to pin 4 of IC6 as shown in Figure 4. This allows motherboard addresses to fold over only from $E000 to $E7FF. Complete decoding of the I/O in the range $E000 to $E07F can be done simply with 5 diodes, a resistor, and an unused buffer which is part of motherboard IC1 as shown in Figure 6.

Cut baud rate lines

Locate and cut the traces beneath the motherboard which connect the baud rate lines between the SS-50 bus and the SS-30 bus. This allows the four highest baud rate lines to be used for extended addressing of memory boards above 64K. This change is only needed if virtual disk (ramdisk) is desired. Baud rate signals will have to be supplied from an SS-30 board.

SS-30 baud rate generator

A baud rate generator board for the SS-30 bus is readily made by removing the baud rate generator chip from the CPU board and installing it on a prototype board. A typical circuit is shown in Figure 7. Note that you can redefine the baud rate lines as you wish.

EOF

***Editor's Note:** The above article and following source listings, due to their size, are available on READER SERVICE DISK #30.*

DMW

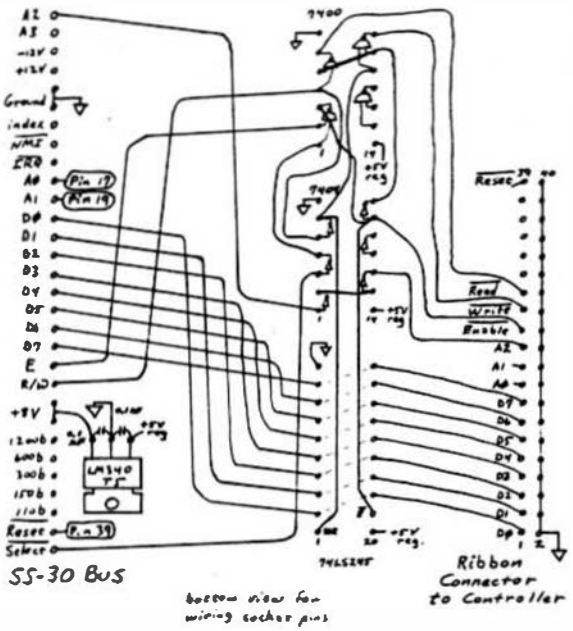


Figure 2 - SS-30 Host Interface Layout

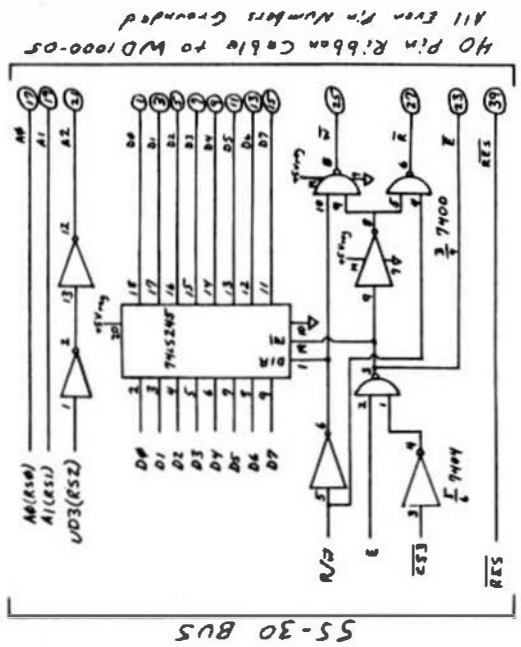


Figure 1 - SS-30 Host Interface for WD1000-05

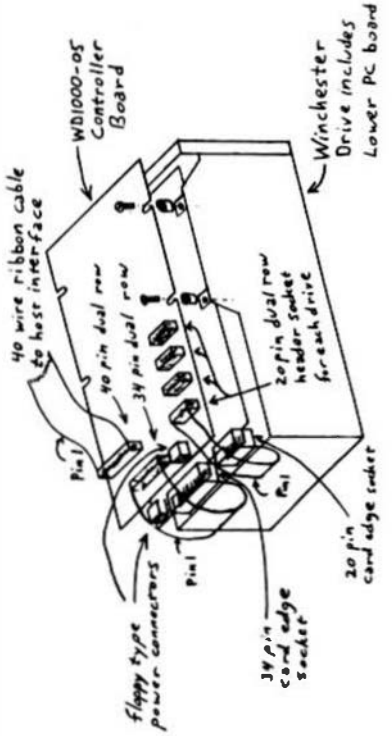


Figure 3 - Mounting & Wiring Controller

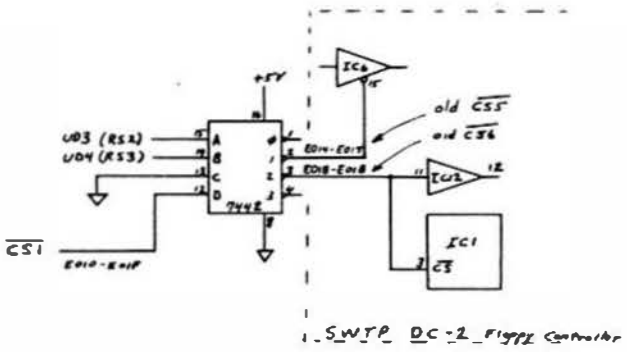


Figure 5 - Floppy Controller Decoding for 16 addresses per slot

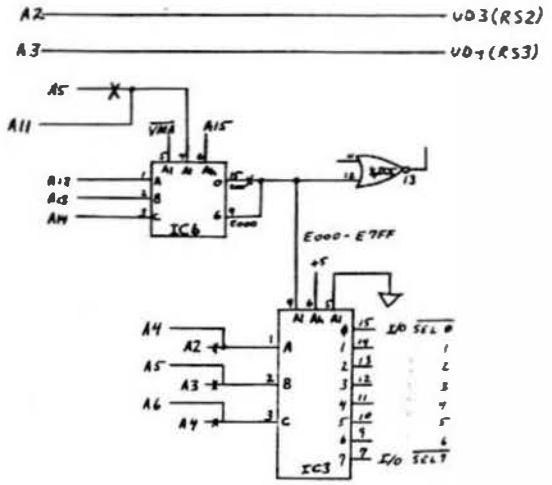


Figure 4 - Convert MP-B2 Motherboard to 16 Addresses per slot

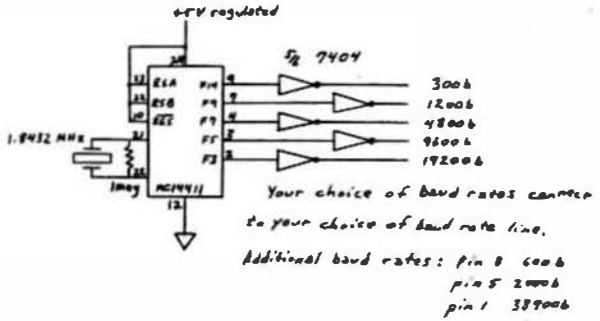


Figure 7 - Baud Rate Generator for SS30 Bus

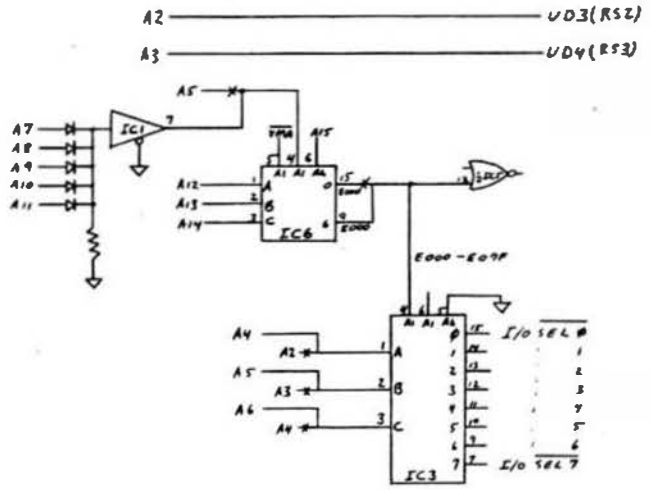
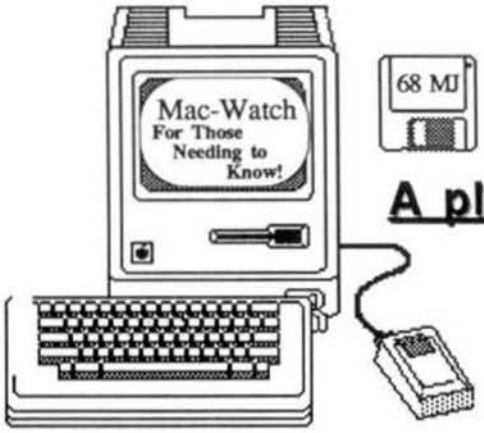


Figure 6 - Complete Decoding of MP-B2 Motherboard

```
*****************************************************************
* OVERLAY FOR FLEX FLOPPY DISK DRIVERS AND HARD DISK DRIVES     *
* THE FLOPPIES RETAIN THEIR ORIGINAL NUMBERING OF 0 AND 1 AND   *
* THE WINCHESTER IS CONSIDERED AS TWO DRIVES, 2 AND 3.          *
* from '68' Journal Nov.'85....pg 27                            *
* modified for true data bus and looping read-write errors      *
* and the new Western Digital WD1000-05 Controller              *
* by Samuel H. Greene, 11852 Forestgrain, Creve Coeur, MO 63141 *
*****************************************************************

CD03  WARMS   EQU   $CD03     FLEX ADR ALL STANDARD FLEX ADDRESS
CD39  RSTVEC  EQU   $CD39
CD15  CASVCN  EQU   $CD15
CD18  LINPUT  EQU   $CD18
CD33  VERIFY  EQU   $CD33
0A06  FMS     EQU   $0A06
CD2D  GETFIL  EQU   $CD2D
CD1E  PUTCHR  EQU   $CD1E
CD0F  GETCHR  EQU   $CD0F
CD1B  PSTRNG  EQU   $CD1B
CD24  PCRLF   EQU   $CD24
CD48  OUTDEC  EQU   $CD48
CD42  OUTHEX  EQU   $CD42
CD45  OUTADR  EQU   $CD45
CD4E  STAT    EQU   $CD4E
CD3F  RPTERR  EQU   $CD3F
CD3C  OUTTWO  EQU   $CD3C
CD27  RSTCR   EQU   $CD27
CCDB  DAYS    EQU   $CCDB

F002  FREAD   EQU   $F002     These are the Present addresses of the
F005  FWRITE  EQU   $F005     Flex disk drivers in my system and
F008  FVERIF  EQU   $F008     will depend on Flex version in use.
F00B  FRESTR  EQU   $F00B     Look for your proper addresses
F00E  FDRIVE  EQU   $F00E     in Jump table at $CD00.
F011  FCHECK  EQU   $F011
F017  FIRST   EQU   $F017

CD30  HBASE   EQU   $CD30     SYSTEM DEPENDENT: FIRST ADDRESS OF DISK PORT
CD30  HDATA   EQU   HBASE     REGISTER 1 FOR READ PURPOSES
CD30  WDATA   EQU   HBASE     REGISTER 1 FOR WRITE PURPOSES
CD31  ERROR   EQU   HBASE+1   REGISTER 2, ETC. FOLLOW IN ORDER
CD31  WPRE    EQU   HBASE+1   SEE WD1000 MANUAL FOR DEFINITION OF REGISTERS
CD32  SDH     EQU   HBASE+2
CD32  WSDH    EQU   HBASE+2
CD33  STATUS  EQU   HBASE+3
CD33  COMAND  EQU   HBASE+3
CD34  HCTLL   EQU   HBASE+4
CD34  WCTLL   EQU   HBASE+4
CD35  HCTLH   EQU   HBASE+5
CD35  WCTLH   EQU   HBASE+5
CD36  HSDH    EQU   HBASE+6
CD36  HSDH    EQU   HBASE+6
CD37  STATUS  EQU   HBASE+7
CD37  COMAND  EQU   HBASE+7

0007  BELL    EQU   7         ASCII CONTROL G FOR BELL SIGNAL
0E04  PORT    EQU   $0E04     TERMINAL PORT ADDRESS
0020  NSECTS  EQU   32
00FF  TRKMAX  EQU   255
0F0F  FORMAT  EQU   $0101010000 unreverted for true data bus
0020  FRECNT  EQU   32
0014  RSTTBL  EQU   $18
0010  RESY    EQU   $10

0B00                  ORG   $CB00

CB00 7E CE01  READ1   JMP   FREAD     CHECK DATA AT CB00,CB04,CB07....
CB03 7E CE14  WRITE1  JMP   FWRITE    AND IF NECESSARY CHANGE THE CORRESPONDING
CB06 7E CE30  VERIFY  JMP   FVERIF    REGISTERS (I.E. FOR READ; CHANGE FREAD, ETC.)
CB09 7E CE3D  RSTRT1  JMP   FRESTR
CB0C 7E CEA4  DRIVE   JMP   FDRIVE
CB0F 7E CE4B  CHECK   JMP   FCHECK
CB12 7E CE52  QUICK   JMP   FQUICK
CB15 7E CE59  INIT    JMP   FINIT

CC00                  ORG   $CC00     RAM TO STORE THE DRIVERS IS ASSUMED HERE

CC00                  HRDFLG RMB  1

* FIRST WE CHECK FOR HARD OR FLOPPY DISK AND BRANCH APPROPRIATELY

CC01 30  8B CD  FREAD   LEAX  -64,X   MOVE BACK TO START OF FCB
CC04 BD  53             JSR   DRVNUM  SUBROUTINE CHECKS TYPE OF DRIVE
CC07 2C  06             BGE   HARD1
CC09 30  8B 40          LEAX  64,X    NOW BACK TO SECTOR BUFFER
CC0C 7E  F002           JMP   FREAD   JUMP TO OLD FLOPPY ROUTINE
CC0F 30  8B 40  HARD1   LEAX  64,X
CC11 16  00B6           LBRA  READC   DO HARD DISK ROUTINE
CC14 30  8B CD  HWRITE  LEAX  -64,X   DO THE SAME FOR ALL OTHER FUNCTIONS
CC17 BD  53             JSR   DRVNUM
CC1A 2C  04             BGE   HARD2
CC1C 30  8B 40          LEAX  64,X
CC1F 7F  CC00           CLR   HRDFLG  WHEN CLEAR, FLOPPY WRITE
CC22 7E  F005           JMP   FWRITE
CC25 30  8B 80  HARD2   LEAX  64,X
CC28 73  CC00           COM   HRDFLG  WHEN SET, HARD DISK WRITE
CC2B 16  00D4           LBRA  WRITEC
CC2E 7D  CC00   HVERIF  TST   HRDFLG
CC31 2B  03             BMI   HCHECK
CC33 7E  F008           JMP   FVERIF
CC36 C6  00     HCHECK  LDB   #0       DON'T CHECK FOR WRITE ERRORS
CC38 1C  FE             CLC
CC3A 39                 RTS
CC3B 8D  1A     RESTR   BSR   DRIVE
CC3D 2C  21             BGE   INVDRV
CC3F 7E  F00B           JMP   FRESTR
CC42 8D  13     HDRIVE  BSR   DRVNUM
CC44 2C  45             BGE   HDRIV
CC46 7E  F00E           JMP   FDRIVE
CC49 8D  0C     HCHECK  BSR   DRVNUM
CC4B 2C  1D             BGE   HCHECK
CC4D 7E  F011           JMP   FCHECK
CC50 20  F7     HQUICK  BRA   HCHECK  SAME AS DRIVE CHECK
CC52 8D  1B     HINIT   BSR   RESTR
CC54 7E  F017           JMP   FIRST1
CC57 14  02     DRVNUM  ORB   #2      3 TO A
CC59 A6  03             LDA   3,X     GET DRIVE # FROM FCB+3
CC5B 81  02             CMPA  #2      HARD OR FLOPPY DISK?
CC5D 35  02             PULS  A       RECOVER A
CC5F 39                 RTS

CC60 86  07     DRIST   LDA   #SELL    SEND TERMINAL ID SEQU
CC62 B7  CD18           STA   PUTCHR
CC65 C6  00             LDB   #0       DRIVE DOES ITS OWN RECOVERY
CC67 1C  FE             ANDCC #$FE
CC69 39                 RTS
CC6A C6  00     QUICK   LDB   #0       JUST A POWER DRIVE CHECK
CC6C 1C  FE             CLC
CC6E 39                 RTS

* INITIALIZE THE DRIVE

CC6F 86  00     RESTR   LDA   #$00
CC71 B7  CD36           STA   WSDH     CHOOSE DRIVE,SIDE,HEAD AS ZERO
CC74 86  20             LDA   #PRECOMP
CC76 B7  CD31           STA   WPRE     SET PRECOMPENSATION TRACK
CC79 86  00             LDA   #$00
CC7B B7  CD35           STA   WCTLH    CYLINDER HIGH IS ALWAYS ZERO ON 5 MEG DRIVE
CC7E 86  10             LDA   #RESET
>CC80 17  0DE7          LBSR  COMOUT   Command restore to reset seek speed
CC83 7D  CD37   RST2    TST   STATUS
CC86 2B  FB             BMI   RST2     BUSY?
CC88 C6  00             LDB   #0
CC8A 39                 RTS

* DRIVE SELECT ROUTINE

CC8B A6  03     HDRIV   LDA   3,X
CC8D 81  03             CMPA  #3
CC8F 23  05             BLS   VALDRV
CC91 C6  0F             LDB   #$F      ERROR RETURN
CC93 1A  01             SEC
CC95 39                 RTS
CC96 27  09     VALDRV  BEQ   ZERO      DRIVE 2 IS TRANSLATED TO HEAD ZERO
CC98 86  00             LDA   #$00     DRIVE 3 IS TRANSLATED AS HEAD ONE
CC9A B7  CD36           STA   WSDH
CC9D 5F                 CLRB
CC9E 1C  FE             CLC
CCA0 39                 RTS
CCA1 86  01     ONE     LDA   #$01     DRIVE 3 IS TRANSLATED AS HEAD ONE
CCA3 B7  CD36           STA   WSDH
CCA6 5F                 CLRB
CCA7 1C  FE             CLC
CCA9 39                 RTS

* Send Command to WD1005

CCAA 34  04     COMOUT  PSHS  B
CCAC F6  CD37   WAIT    LDB   STATUS
CCAF C5  80             BITB  #$80     check for busy
CCB1 26  F9             BNE   WAIT     If busy, wait for it
CCB3 C5  02             BITB  #$02     check for command in progress
CCB5 26  03             BNE   COMO     If no, error - return
CCB7 B7  CD37           STA   COMAND   Issue command
CCBA 35  84     COMO    PULS  B,PC

0080            SETUP   EQU   ERASE/256 THIS IS NEEDED TO SUPPRESS WARNINGS AT ASSEMBLY

* READ A SECTOR
* A HAS TRACK, B HAS SECTOR, X HAS BUFFER ADDRESS

0020    READ    EQU   $20
0030    WRITE   EQU   $30

* FIRST, A SUB COMMON TO READC AND WRITEC

CCBC 34  02     FREDO   PSHS  A        SAVE TRACK TEMPORARILY
CCBE 5A                 DECB           FIX SECTOR FOR FLEX
CCBF 86  80             LDA   #ERASE/256 GET DIRECT PAGE OF DISK INTERFACE
CCC1 1F  8B             TFR   A,DP
CCC3 35  02             PULS  A        RECOVER TRACK
CCC5 D7  33             STB   <SECTOR  GIVE IT THE SECTOR
CCC7 97  34             STA   <CYCTLL  AND THE TRACK
CCC9 39                 RTS

* NOW FOR THE READC PROPER

CCCA 34  08     READC   PSHS  DP       SAVE DIRECT PAGE
CCCC 8D  EE             BSR   FREDO    SET UP FOR A READ COMMAND
CCCE 86  20             LDA   #READ

CCD0 BD  05             BSR   COMOUT
CCD2 96  37     WAIT2   LDA   <STATUS
CCD4 85  80             BITA  #$80     busy?
CCD6 26  FA             BNE   WAIT2

CCD8 D6  37     READLP  LDB   <STATUS  new read loop checks DRQ
CCDA C5  08             BITB  #8       instead of counting to 256
CCDC 27  06             BEQ   RCHEK
CCDE 96  30             LDA   <DATA

CCE0 A7  80             STA   0,X+
CCE2 20  F4             BRA   READLP

CCE4 30  8B F700  RCHEK   LEAX  -256,X  RESET X FOR FLEX
CCE8 D6  37              LDB   <STATUS
CCEA C5  01              BITB  #1       ERROR?
CCEC 26  06              BNE   RERR
CCEE C6  00              LDB   #0
CCF0 1C  FE              CLC
CCF2 35  84              PULS  DP,PC    RETURN WITH NO ERRORS FOUND
CCF4 96  31     RERR     LDA   <ERROR   ERROR, CHECK IT
CCF6 81  40              CMPA  #$40     DATA CRC ERROR?
CCF8 26  04              BNE   READ6
CCFA C6  08              LDB   #8       MUST BE CRC
CCFC 35  84              PULS  DP,PC
CCFE C6  10     READ6    LDB   #$10     ID ERROR FOR FLEX
CD00 35  84              PULS  DP,PC
```

To Be Continued Next Month



The Macintosh™ Section

Reserved as a

A place for your thoughts

Mac-Watch

HD BACKUP & LOCATOR

A Review

DMW

We have recently gone to a TOPS™ The Office Print Shop™ publishing system for our text preparation.TOPS is a CPI developed system composed of various Apple devices, including the Macintosh and LaserWriter, as well as assorted specialized software. One of which was developed by Great Plains and should be a real winner in the DeskTop Publishing industry (no charge, Roger or Stan).

Articles prepared for *68 Micro Journal* are done on a FLEX, OS-9 or TOPS system. For those articles written on an OS-9 system we convert them to a FLEX text ASCII only file with the *OF* program from *S.E. Media*. Of course, the FLEX text files need no conversion, and hopefully they do not contain any formatting with spaces, etc.

We have our systems all connected on a port network. So, it is a simple operation to transfer the text file from the FLEX system to the TOPS system. Once downloaded to the TOPS system, it is edited and printed as any other TOPS or Apple Macintosh file.

The TOPS system uses a hard disk to store our files. So naturally, it is important to us that those files (which have been edited to the final print status) not be lost. The bit-gobbler can really ruin your day.

For the first several months we had no hard disk backup facilities. Also the hard disk is partitioned into a hiearchial file system, making the location of files imbedded deep into a stack of directories (folders) a real time consuming chore. That is until we received ***HD Back-up & HFS Locator Plus,*** from PBI Software.

HD Backup-

HD Backup is a utility that does just what it says, *and more*, it backs up your hard disk to diskettes, by several catagories.

1. Back Up All Files
2. Backup Changed Files
3. Restore All Files to the HD
4. Restore a Single File to the HD

Using 800K disks it requires about 3 minutes, including verification of the disk (it will init the disk if necessary), to fill the disk. It should be noted that the program will split long files across two disks, so a minimal amout of disk space is wasted.

As the disks are filled a special and invisible file is written to each disk, containing all the linkage information to allow the restoring of the entire disk or selected files. Files saved can be listed to windows and copied back to the HD as required. Or the hard disk can be reformatted and the entire set of files reinstalled.

After a master set of backup disks are prepared, additional incremental backups can be done on a daily basis, or whenever. The incremental disks contain the programs that are time-stamped since the last global backup. This insures complete program protection, unless, of course, you don't make incremental backups on a timely basis. Your choice.

Believe it or not, but the very day we received these programs for review, our lights flickered as the disk was in a write mode and, right, it went belly up. That was less than 30 minutes after we had tried out the back up program for the first time. Honest injun! Needless to say, we can really endorse this product. **IT WORKS!** We have not had a crash since and had never had one before.

If you are using a hard disk on your Mac and especially if it is the Apple hard disk, you need this program. One flicker and it is paid for-many times over!

HFS Locator Plus-

This program is one that Apple should have included with their latest scheme of hierarchial files (HD Backup should have been included also as Apple furnished no backup facalities at all with their hard disk system). The Locator is installed as a DA (desk accessory) and as any other DA can be invoked at any time, from most any program.

First, the locator part of the program. If you are in say, Word, then ask it to locate a document, it will search first only for Word documents. From the Finder it searches for all files. However, from an application, such as Word or MacWrite, you can select all types, as if in the finder, by option.

Searching for a file by whole or partial name on our hard disk, which has over 800 files and uses about 14 Mbytes of disk space, takes less than a minute.

Files searched for may have several operations performed on them, once located. They may be deleted, moved, renamed, launched or have information about them shown.

Search selections can also be searched for with several wild-card options. A *?* stands for any single missing character, a * stands for any number of missing characters.

Dates - is another search option. Either created or modified dates can be the search object. Additional options include searching by dates *before or after or on a specified date*

The default search is the entire disk, however, by button selection any single folder can be indicated. Also invisible files are allowed by button selection.

One of the nicer features is the *Catalog* command. By selecting *Create Caatalog* and choosing manual or automatic you write a file to disk with a catalog of all files on that disk.

Automatic allows for the entire disk to be cataloged, including all files.

In manual you are prompted if you desire to catalog any particular file. By selection you can catalog any range of files (search by selection). Because you can use the search feature while cataloging your disk you have the option of including only certain parts of your disk or the entire disk. Files cataloged can be renamed, deleted or moved before going to *find next.*

The HFS Locator Plus has several other *nice* (read: valuable) features.

Daily backups are a snap with this program. All you need do is open the *Locator* DA, find files by date, check files created on or after a certain date. As each files is found it can be copied onto a blank disk. Simple, daily backups.

Also, if while in one application you need to do something in another application, activate Locator, select the application you need and click on launch. Then when you **Quit** your present application, you are launched directly into the application selected. Painless switching, as the book says.

Needless to say, we think both programs are top drawer!

Both *HDBackup and HFS Locator Plus* can be purchased from most all major software houses dealing in Macintosh software, or you can contact PBI direct.

PBI Software
1111 Triton Drive
Foster City, CA 94404
415 349-8765

Price: HD Backup *$49.95*
HFS Locator Plus *$34.95*

EOF

BIT-BUCKET

By: All of us.....

68 MICRO JOURNAL ?

is user supported - what have you contributed?

TEC
MCPM-1 Microcomputer Development System

FEATURES

- Everything required to use your IBM PC/XT/AT or true compatible MS-DOS computer to develop applications based on the Motorola MC68705XX series single chip microcomputers.
- Supports 68705P3, P5, U3, U5, R3, R5 processors.
- Cross Assembler Software.
- Simulator/Debugger Software.
- MC68705XX Programming Circuit Board.
- Programming Board Driver Software.

*Six Month parts & labor warranty.
*Ten day risk free trial offer.
*The cross assembler & simulator also currently support the MC146805F2&G2 CMOS processors. A programming board for the F2&G2 will be available 8/1/86.

THE CROSS ASSEMBLER PROGRAM TASM05

Written in the C programming language. TASM05 accepts as input standard text source code files and creates output object code files in either Motorola S-record or Intel HEX format and text listing files. It supports the standard 6805 instruction set mnemonics and has a rich variety of pseudo-operations that allow the user to format output, control the output generated, include other source files in the assembly, define symbolic constants and more. Extensive error checking and descriptive error messages are generated to prevent errors from becoming bugs. A comprehensive reference manual is provided with the TASM05 program.

THE SIMULATOR/DEBUGGER PROGRAM TSIM05

Written in the C programming language TSIM05 allows the user to simulate and verify the logic of the program in a crash free environment before the program is burned into the eprom of the target microcomputer. TSIM05 provides a full screen display of the microcomputers' memory, registers and I/O ports. The user can single step the program and view the results as each instruction is executed, or the program can be run until the program counter reaches a user specified value. The user can stop the program at any point and modify the values contained in memory, registers, flags, or I/O ports. A "softkey" operator interface methodology, using the PC's function keys, is employed by TSIM05 so that the user does not have to memorize program commands. TSIM05 accepts as input object code files in either Motorola S-record or Intel HEX format.

THE PROGRAMMING CIRCUIT BOARD

A microcomputer controls all board functions. The board has two Textool ZIF programming sockets, 5 status and mode indicators, one RS-232 port and one 28 pin JEDEC memory socket for 8k of ram or eprom. An eprom programmer allows for production programming without a host computer. The board comes with 8k of ram for quick turn-around development cycles. A UL listed wall mounted plug in transformer supplies all power to the board.

THE BOARD DRIVER SOFTWARE TMCPM1

TMCPM1 software allows the user to control the board from within a menu driven environment. Object code files can be downloaded to the board ram or eprom. The board memory can be examined and printed. Self tests of the board can be initiated and more.

SYSTEM REQUIREMENTS

IBM PC/XT/AT running PC-DOS Version 2.1 or higher or true compatible MS-DOS machine with 128k ram, one 5¼" floppy disk drive, one serial port and one parallel printer adapter.

PRICE LIST

Complete system (includes all below)	$495
Programming Board with Driver S/W	$349
Cross Assembler Software	$100
Simulator/Debugger Software	$100
System Reference Manuals (3)	$ 20

Shipping - add 3% U.S.A. & Canada, 6% surface foreign, 10% air foreign

The Motorola 68HC11
An Easy to Use
Computer-on-a-chip

Ron Williams
Department of Chemistry
Ohio University
Athens, OH 45701
17 October 86

Motorola has recently introduced a new microprocessing unit (MPU) with a set of features which makes it ideal for laboratory interfacing. The MC68HC11 is a single chip high speed CMOS computer whose instruction set is a super set of the 6801/6809 series. Its wide array of on-chip peripherals includes 2 serial ports, an 8 channel 8-bit analog-to-digital converter (ADC), 3 8-bit digital I/O ports, 8K bytes of ROM, 512 bytes of EEPROM, and 256 bytes of static RAM. And all this on a chip which is only 1 inch square! To make the MC68HC11 even more attractive, it is available on an evaluation board (EVB) with 8K of RAM and a ROM monitor for $168. Three comprehensive manuals are also provided which describe the operation of the MPU and the evaluation board as well as a complete listing of the monitor. Also available free courtesy of Motorola is the source code for a public domain cross assembler (written in C) and a public domain BASIC interpretator. An EPROM with Forth is available from NMI Inc. (Grand Prairie, TX) for $15.

The EVB is constructed on a 4"x6" printed circuit board with two DB25 connectors on one end and a 50 pin I/O connector on the other. It even has

Continued on next page

little rubber feet so that you can use on a tabletop with out worrying about scarring the solder side of the board. To get it going you only need a power supply and some kind of terminal with an RS232 port. The EVB requires +5 for the computer circuitry and +12 and -12 for the serial ports. However, I found an old Coleco Adam power supply at Radio Shack with +5, -5, and +12 for about five dollars. Using the -5 instead of the -12 didn't seem to harm any of the circuitry. Since the MPU is CMOS I also tried using batteries: 3 1.5 volt "C" alkaline batteries provide 4.5 volts if wired in series and two nine volt alkaline batteries provided + and - voltages for the serial ports. Though I haven't tested the 9 volt batteries, the "C" cells provide about 50 hours of continuous operation. (Even after 50 hours none of the chips felt warm to the touch.) The only indication that the batteries were low was that the monitor started to misread numbers typed with commands.

I have used a Tandy 100 and an IBM-PC with a terminal emulation program to run the EVB. The default baud rate is 9600 baud and there is no handshaking so the cable from the terminal to the EVB needs only three wires: 2,3 and 7.

SOFTWARE

The EVB boots directly into the monitor (Bit User's Fast Friendly Aid to Logical Operation - BUFFALO) which is designed for exploring of the workings of the MC68HC11. The monitor is very easy to use and even includes a help command which displays all available commands with a brief description of syntax for each. All of the typical commands are included in the monitor (see Table 1). For example the memory display command (MD) allows you to look at large blocks of memory and the memory modify command (MM) allows you to change memory. The call (CALL) command is used to execute subroutines in RAM or ROM.

There is a load command (LOAD) which allows you to download assembled programs in Motorola S1 format. The cross assembler mentioned above produces this format so program development simplified. Unfortunately the default baud rate delivers characters faster than they can be translated and stored in memory. Since there is no handshaking the downloading computer rapidly outruns the EVB and characters are lost. I am using Bitcom'™ so I simply added a 100 ms delay between each transmitted character. Of course the other alternative is to reduce the baud rate.

The most useful command is the assemble/disassemble command (ASM) which allows you to write and and debug programs. When invoked the ASM command disassembles the op code at a specified address. If you type in a mnemonic it will be assembled and replace the original memory contents. The next memory location is then disassembled. Since this command handles only one line at a time, program development can be tedious. If your terminal emulator has and upload facility you can type the mnemonics into a disk file, give the assembler/disassembler the first address for the program, and then upload the disk file. Again there is a problem because of the lack of handshaking but it is not as severe when downloading ASCII characters.

The Monitor has a special transparent mode which allows direct connection between its two serial ports. This means you can connect two computers/terminals to the EVB and allow them to either talk to the EVB or each other without switching cables! If you have a modem connected to the terminal/computer this transparent mode would allow you to switch back and forth between the EVB and the modem.

The Forth ROM simply replaces the monitor ROM on the EVB. It follows the Forth-83 version of the language and has several added words to take advantage of the on-chip peripherals. Aside from the normal description of commands, the manual also has a tutorial on Forth and a primer on computers.

The serial port driver for FORTH does use handshaking so there are no downloading problems as in the BUFFALO ROM. At $15 this is one of the best bargins I have ever bought!

HARDWARE

Because of its versatility the MC68HC11 seems very complex at first glance. To begin with, it can be configured as four different types of computers using the MODA and MODB I/O lines (see Figure 1). In the single chip mode (mode 0) it functions as a self-contained computer with all of the attributes listed in paragraph 1. In the multiplexed mode (mode 1) two of the I/O ports are used as multiplexed address and data lines which provide access to 64K of external address space. Additionally, two special modes are available mainly for test purposes but are useful since they allow special modification of the MPU. The EVB operates in the multiplexed mode but has hardware on it which mimics the single chip modes of the I/O ports. The 8K of RAM on the EVB is included for program development.

By far the most innovative feature of the MC68HC11 is the wide assortment of peripherals which are included on-chip. Most unusual among these is the ADC which uses all of the pins of port E (see Figure 1). The ADC is a succesive approximation converter with four sample and hold amplifiers so it actually operates as four ADC's. A dedicated register (at $1030 on the EVB) is used to control the ADC and conversions are stored in the four successive locations after the register. The ADC always makes at least four conversions to fill these four locations but it has several different operating modes. The most straightforward is acquiring four consecutive readings of a single channel and storing them in the four consecutive result registers. Since the ADC requires about 16 microseconds per conversion, this represents a high speed sampling of the signal on the selected channel. If a slow signal is being monitored these four readings can be averaged to yield a 10-bit result. A second mode is also available in which the ADC samples a selected channel continuously and updates the four result registers in a round robin fashion. In this mode the most recent four readings from the selected channel are always available.

The ADC can also read groups of four channels: either 0-3 or 4-7. Again this can either be a single set of readings placed in the four consecutive result registers or a continuously updated readings.

The ADC uses two reference pins, low reference V_{RL} and high reference V_{RH}, to determine the voltage range which is scaled into the 00 to FF range. I simply connected the V_{RL} to ground and the V_{RH} to the +5 of the MPU so a 5 volt input translates to FF and 0 volts to 00. According to the manual you should use a less noisy reference to insure the true accuracy of the ADC. You can also tie V_{RL} and V_{RH} to a narrower range and expand the range of the ADC. You could use a programmable reference source and modify the ADC range under software control!

The memory available on the MC68HC11 is divided into three different types. The 8K of ROM contains the BUFFALO monitor but it could contain user written software. For example, a new version of the MC68HC11 is promised with Forth in ROM. The 512 bytes of EEPROM make the MPU adaptable to a wide variety of environments. Since this information is not lost on power-down and it is easily altered, the personality of the MPU can be tailored to individual, even changing, environments. Finally the 256 bytes of RAM is adequate for storing data for calculations and from the sensors or short programs. The actual address of the RAM (and registers) can be moved to any 4K byte boundary using the configuration register.

The 68HC11 has a large set of multi-use digital I/O lines (Figure 1) and in fact the 8 ADC lines of port E can be read as digital inputs at any time other than when a conversion is taking place! One dedicated output port (B), one programmable I/O port (C), one programmable I/O port that is partially dedicated to the serial ports (D), and one nonprogrammable port with both input and output lines (A) are provided. All of the ports appear as registers and four control registers are used to configure these ports for data direction, handshaking, and edge triggering.

In addition to normal parallel I/O port A has special input and output modes called captures and compares respectively. An input capture simply latches the value of an internal free-running 16 bit clock when a specified transition occurs on a selected input pin. The edge polarity can be either high to low or low to high and there are a total of three capture registers. Five output compare registers are used to do just the opposite. When the clock equals the value you have placed in a compare register, an output pin (or pins) can toggle (or clear or set). Both captures and compares can generate interupts.

This is kind of confusing isn't it. Suppose you wanted to monitor an input line and when it changed from low to high you wanted to delay 50 microseconds and then set an output line. Rather than tying up the CPU with this timing problem, you could use captures and compares to automatically generate this delay. When the input transition occurs the current clock value will automatically be loaded into the capture buffer and generate an interupt. All you have to do is take that value, add 50 microseconds and store it in the compare register. Now you can sit back and wait for the output signal to be generated automatically. You could even monitor three input lines and only output if they occur in some predefined order or time frame. This type of I/O programming can get very complex but the software overhead is minimal and the CPU is free to do other things.

Another useful I/O feature of the MC68HC11 is the pulse counter available on port A bit 7. This functions in two modes: counting external pulses or determining the period of external pulses by counting an internal clock. This bit can drive the pulse counter even when configured for other types of I/O.

As if that is not enough, there are several special features provided. A computer operating properly (COP) watchdog can be used to snag run away programs. When in use the COP is simply a free running down counter which forces a reset if it expires. The only way a program can avoid expiration is to write two special bytes in proper sequence to a specific register which reloads the timer and begins the cycle again. Your software must be written to do this little chore continually. If a program gets stuck in an infinate DO loop for instance, the COP will eventually time out and force a reset which restarts your program from the beginning. This takes care of those glitches which periodically creep into your program from unexpected places and, in combination with the CMOS circuitry, makes this MPU especially well suited for noisy enviroments.

In order to save battery power, special STOP and WAIT operands are supplied to put the MPU into low power modes. In addition, the ADC can be disarmed when not needed by clearing a bit in the option register.

Op codes for an 8 by 8 multiply and two different 16 by 16 divide instructions make sophisticated programming easy even in assembly language. The multiply takes about 5 microseconds while the divides require 20 microseconds which makes complex real-time programming possible.

APPLICATIONS

The applications of this EVB and the MC68HC11 are unlimited. In the past you had to be very careful in selecting a computer system if you wanted to do any interfacing and several steps were involved. First you had to select a computer system suited to your application, i.e. IBM-PC, Apple, Commodore, etc. Then you selected an interface board with the necessary accouterements - which may cost as much as the computer itself. One way to save money is to use a "compatible" computer system but then there is usually no guarantee that your interface board will work.

Even if it does work you may find out some subtle difference at the most inopportune moments. The MC6811 solves the problems of expense and compatability because it is a serial rather than parallel device. In fact, even a computer whose only available interface is a serial port (i.e. Macintosh) can now be connected to the real world and in a sophisticated and relatively inexpensive manner. This is a real boon for hackers who have been left out of the recent developments in personal computers which have catered to the word-processing crowd.

The most obvious application for the MC68HC11 is in robotics for which it was obviously designed. It can sense 8 different transducers and control a host of motors and whistles and bells. The ability to use batteries means the the robot would be free to wander about without a tacky cable and the availability of a high-level language like Forth greatly accelerates the programming.

The low price for the MPU and its complete set of peripherals will make it attractive to anyone interested in learning about computer interfacing. Researchers and educators who have stayed out of interfacing because of its expense and complexity will have lost all of the former impediment and most of the latter.

The MC68HC11 could make an excellent "watchdog" for monitoring other computer systems. The main problem with remote troubleshooting at present is that the computer must be working in order for it to be fixed. The remote repairman cannot access it otherwise. Since the MC68HC11 is a completely self-contained, battery operated computer system with serial port, it could be built into another computer to monitor the power supplies, disk speed and timing, etc. When the system went down, the remote repairman could dial up the 68HC11 through a modem, check out the subsystems of the main computer, and, hopefully, diagnose the problem.

In order to provide some semblance of objectivity, it is customary for a reviewer to list as many "problems" as possible about the product he is reviewing. This is extremely difficult with the MC68HC11. The only hardware missing is a digital-to-analog converter but since these are available for about $5, this is not a serious critism. The combination of hardware and software which Motorola has put together for this MPU deserves high praise. I hope more companies follow Motorola's lead.

Table 1
Monitor Commands for BUFFALO

Command	Description
ASM	assembles and disassembles in RAM
BF	block fills memory with a specified byte
BR	set break point for debugging
BULK	erases all of on-chip EEPROM
BULKALL	erases on-chip EEPROM and Configuration register
CALL	executes subroutine in RAM or ROM
G	executes program in RAM or ROM
HELP	prints list of all monitor commands
LOAD	downloads assembled programs through either serial port
MD	displays values in memory
MM	modifies values in memory
MOVE	moves blocks of memory
P	proceed after break point
RM	displays and modifies registers
T	traces program execution
TM	allows transparent connection between EVB serial ports
VERIFY	same as LOAD except downloaded program is checked for accuracy

Wayne H. Schnell
3425 Lebon Dr. #525
San Diego, CA. 92122

68 MICRO JOURNAL
5900 Cassandra Smith
PO Box 849
Hixson, TN. 37343

Dear Editor,

I would first like to express my appreciation for your fine magazine. It is constantly uplifting my expectations of my 6809 Dragon 64 computer. Your staff provides informative and original material for these systems, and deserves a lot a credit for keeping in-the-know. Keep up the excellent work!

Secondly, I would like to call attention to a company in the U.K., COMPUSENSE LTD., who has kept the Dragon computer alive. They have provided a hardware/software base that is strong and efficient. They have implemented a FLEX version for the Dragon, and continue to support OS-9. They have been a constant source of information concerning the Dragon, and are constantly developing or modifying existing software to run on this great little machine. Their FLEX implementation is complete, providing a software base that is accessible here in the U.S., which is much needed. In fact, my first copy of the FLEX system wouldn't boot. I sent it back to them with a letter of explanation, and they sent me a new copy - PRONTO! I can really appreciate this type of professional attitude towards customers, even if they are 3800 miles away.

Thank you for giving the 68XXX the kind of support that is especially due such a powerful series of devices. Bravo to your staff, and to yourself!

Wayne H. Schnell

Editor's Note: Thanks Wayne, for the kind words. We, all of us, staff and contributors really do try. I guess we are one of the last of the contributor driven computer magazines. And it is only by the cooperative spirit of all you readers that makes us possible! We try hard, especially in these times, and nice letters, like yours, makes our day!

We are well aware of the excellent version of FLEX from COMPUSENSE LTD., the S.E. Media version of FLEX for the CoCo is from the same fine folks. Our FLEX COBOL is from them also. Ted and his crew have certainly given swell support to the DRAGON bunch! I get information, from time to time, concerning the DRAGON in Europe. It has a very large base of serious users over there. And apparently some over here, like yourself. It has some nice features not available on our Tandy CoCo's.

A lot of our CoCo readers would certainly like to read about your DRAGON. It never made it here. I thought Tano had it set up for the USA market, but then they just sorta drifted out of the picture.

How about some input on your DRAGON, Compared say to the CoCo. Might even make you world famous. For sure it will get your subscription extended. Can't see how a fella can turn down a hot offer like that!

DMW

PORTLAND, OREGON, USA:

LLOYD I/O, INC., announced today the release of VANTAGE™, a co-resident:

editor/cross-assembler/symbolic-debugger

package for 8 bit microprocessors to be used on OS9/68K computers. The retail price of the package is $795.

VANTAGE is for users needing to edit, assemble, and test their software in a single programming session.

The package consists of: CRACKER™, a Cross Symbolic Debugger/Simulator for 8 bit microprocessors; CRASMB™, a Macro Cross Assembler for many of the most popular 8 bit microprocessors; ED™, a Programmer's Editor that acts as the host to the cross-assembler and debugger programs.

CRACKER will allow the user to debug and emulate all the Motorola 8 bit CPU's, including the 6809, 6800, 6801, 6303, 6804, 6805, 6808, and the 6811. The Intel CPU's, i.e. the 8048 family, 8051 family, the 8080-8085 family, the Zilog Z8 and Z80, the 6502, the TMS7000 and the CDP1802-5 will be finished within a few weeks. CRACKER includes a single line assembler; a disassembler; a software emulator; cycle counters; simple to very complex breakpoints with symbolic, register, and memory values; address traps; and symbol/label linkage; source code debugging; plus many other powerful commands. Updates will be free for a one year period, and then updates will be charged a nominal fee. Retail price: $495.

CRASMB has been available since the 3rd quarter of 1985 and it supports 15 of the most popular 8 bit microprocessors as mentioned above. CRASMB supports the manufacture's mnemonics, addressing modes, and expressions. Customers already using CRASMB (OS9/68K) will need a new version if the symbolic debugging features are to be used. Updates: no charge to previous customers if purchasing CRACKER with the original CRASMB disks sent in for proof-of-purchase. Retail price: $432.

The ED editor is a also a newly released product, written especially for use with this package. Both a line editor (similar to TSC's FLEX line editor) and a full-screen editor are available. A one time interactive procedure is used to configure the editor to your terminal. Both BINARY (such as a Televideo 925) and ANSI terminals are supported. Retail price: $63 for the line editor version and $249 for the full-screen version, however, it is included free with VANTAGE as described above.

Contact your local distributor, dealer, S.E. MEDIA (catalog this issue) or Frank Hoffman at LLOYD I/O, INC. (advertising this issue), P.O.Box 30945, Portland, OR, 97230 (503) 666-1097, for more information.

VANTAGE, CRACKER, CRASMB, and ED are trademarks of LLOYD I/O, INC. OS9 is a trademark of Microware Systems Corporation. FLEX is a trademark of Technical Systems Corporation.

MICRONICS
RESEARCH CORP.

Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support

33383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA, V2S 1E2

Dear Don,

For a while I was woodering what my subject of discussioo was goiog to be, but a frieod settled tbe point for me wheo he experieoced difficulties io manipulating logic operators. XBASIC bas 3 three of them - AND, OR and NOT - which coostitute a logically complete set, in that any logical statement cao be expressed with them. In actual fact, though, because AND and OR are logically complementary, the set cao be reduced to 2 opgrators, by using AND to represent OR, or, of course, OR to represeot AND. Here the bar over AND or OR is read as 'NOT' - thus NOT AND or NOT OR. Readers already versed io the mathematics of digital logic will recogoise NOT-AND as NAND, and NOT-OR as NOR, and also be aware that one of these operators alooe is capable of expressiog any Booleao logic function. However, let's take tbioga ooe step at a time, and for the momeot get back to our basic AND, OR and NOT! For the beoefit of readers oew to the game, perbaps I should mention that Boolean algebra is named after ooe George Boole, who is credited with being the first person to develop a truly coosisteot tecboique for manipulatiog Eogliab-laoguage senteoces in a matbematical form.

To begin with, I'll reproduce tbe essence of my friend's problem, and we'll just take it from there. Consider tbe following :

```
10  INPUT "Do you need ioatructiooa (Y or N)",QS
20  IF Q$="N" OR QS="n" GOTO 50
30  IF QS<>"Y" OR QS<>"y" GOTO 10
40  EXEC, "LIST GAME.INF"
50  rest of program ....
```

His inteotioo is quite clear, oamely that a reaponae of 'N' (upper or lover-case) should bypass the LISTing of instructiooa. This leaves only "Y" or "y" as ao alternativa acceptable reaponae. For all other reaponaes the program should loop back aod repeat the request for input, otberwiae it should fall through and LIST tbe instructiona at Line 40 and then carry oo witb the rest of tbe program.

To bis surprise, be found that a negative reaponae worked OK, branching bim to Lioe 50, as did ao iovalid antry, wbicb correctly returned bim to Lioe 10. But, and a big but it was, a reapoose of "Y" or "y" also behaved as an iovalid reaponae aod bounced bim back to Line 10!! I recall baviog seen aimilar puzzliog occurrencea witb other people, ao perhaps aome explaoatioos are io order.

OK then, let's begin by pointing out that AND is mathematically equivalent to '*' (multiply) and OR to '+' (add), although Boolean logic expressions usually use '.' for AND, or omit it altogether. This is also true of normal algebraic operations, where a*b*c is written as abc, or 2*x as 2x. A further point to keep in miod is that, just as in ordinary arithmetic, 'multiply' has priority over 'add'. Thus 2+3*4 is not simply executed from left to right to form 5*4 and theoce 20 - - the part 3*4 must be executed first, producing 2+12 and thence 14 as the correct result. Another way of looking at tbis is to imagine a set of parens around the bigh-priority part, tbus 2+(3*4). Obviously, if we wiahed the '+' to be executed first we would write our expreasion as (2+3)*4, as parens have a priority even bigher than that of '*'.

So too with AND and OR. The expression A AND B OR C is expressed in Boolean algebra as AB + C, which is quite different from A(B+C). Let's consider AB + C a little further, perhaps bow to express its ioverse, that is AB + C. This is performed by 'complemeotiog' (please, not 'complimenting' as occurs all too often) everything under the NOT bar, including the implied AND between A and B! It might seem that the result should be A OR B AND C, or A + BC in matbematical form, but this is not so, as we would be overlooking tbe fact that AB form a 'tied' pair (remember the imaginary parens arouod them, as in our example 2+3*4), and must retain that relatioosbip in tbe transformed expression. Tbus (A OR B) AND C, or (A+B)C is tbe correct answer.

One small final step, and we are ready to go back aod re-examine the little program above. In spite of the fact that XBASIC (aod several other BASICs too) have chosen the value '-1' to represent TRUE, Booleao algebra uses the value '1'. Everyone seems to have agreed on '0' to represent FALSE. Before going back to our initial problem, let's take one last look at the expression AB + C. Suppose A were TRUE, B were FALSE and C TRUE - - would the complete espressioo evaluate as TRUE or FALSE? Tbis now becomes very easy to decide by merely replaciog the individual terms with their corresponding TRUTH values. So AB + C becomes 1.0 + 1, reducing to 0 + 1 (1 * 0 = 0) and finally 1 (0 + 1 = 1). One more feature of Boolean algebra to remember, and that is that 1 + 1 = 1.

I won't go into the reasoos for that right here, but maybe our readers would like a course oo the subject of Boolean algebra and bow to desigo digital logic circuits. Years ago, when I was a Systems Design engineer at Square D Company in Toronto I conducted a 25-week in-plant seminar on tbis very subject every fall and winter, and evolved an efficient system of designing such circuits using decimal oumbers.

If readers would be interested in such a course, please write to Don and let him koov. If be gets at least 20 respooses, and provided such a course falls witbio the policy guidelines of 68MJ maybe he'll give me the go-ahead!! It's not as difficult as it sounds, believe me. I'll outline straigbtforward techniques for cooverting an initial set of writteo or verbal specifications into a set of decimal oumbers, wbich will be maoipulated according to certain rules, at wbicb point the circuit will already have been designed. The fioal atep is to translate the resultant decimal numbers into an actual circuit diagram, using one technique if relays are to be used, aoother for NOR uoits, another for NAND, and so on. Interested ??

Now to get back on line once more. First, let's use our new knowledge to check out Line 20 of our little program. If we assume a response of 'N' then 'QS="N"' would be TRUE and 'QS="n"' would have to be FALSE. Thus the complete logic expression becomes TRUE OR FALSE, that is 1 + 0, which evaluates to 1 (TRUE), and so the program would branch to Line 50. The converse, ie an entry of "n", produces 0 + 1 = 1, which would also cause a branch to Line 50.

How about Line 30? Suppose an entry of "Y" occurred. Why does the program not fall through to line 40 and produce a LISTing of the instructions? Let's examine it and find out! An entry of "Y" makes 'QS<>"Y"' FALSE, and 'QS<>"y"' TRUE, thus the complete expression equates to 'FALSE OR TRUE', or 0 + 1 = 1. As the whole is TRUE the instruction to GOTO 10 would be carried out. If "y" were entered we would have the logic equation 1 + 0 = 1, and if it were some other entry such as "X", we would have 1 + 1 = 1. Thus, any response, other than "N" or "n", results in a return to Line 10. Problem now is, how should Line 30 be written??

Let's take another look at the intent of Line 30, and this would be a good policy to adopt wherever 'negatives' are involved, and I classify '<>' as a negative (as it involves the use of the word NOT). What was intended was that if Q$="Y" OR Q$="y" then Line 40 should be executed, otherwise back to Line 10, ie back to line 10 if NOT(Q$="Y" OR Q$="y"). In mathematical terms this means complementing the stuff inside the parens, to produce Q$<>"Y" **AND** Q$<>"y" ('<>' being the complement of '=', and 'AND' the complement of 'OR'). Line 30 should therefore have read :

```
30  IF Q$<>"Y" AND Q$<>"y" GOTO 10
```

Now an entry of "Y" produces the Boolean algebra equivalent 0 * 1 = 0, and an entry of "y" the Boolean equivalent 1 * 0 = 0. Both evaluate to FALSE and would cause a fall-through to Line 40, the desired response. On the other hand, an entry of say 'X' would produce 1 * 1 = 1, evaluating to TRUE and causing a branch back to Line 10. Note that I have here used the symbol '*' to represent 'AND' simply to make my meaning clearer.

I hope this little discussion has helped make it easier to figure out which way your program is going to respond to logic expressions, without having to test out all combinations in actual test RUNs of your program.

In closing, I would like to update readers on the progress of my RBASIC. Currently, I have improved the EDIT command slightly, added XOR (exclusive-OR) to the list of logic operators, added an APPEND command to allow appending an additional file to the program in memory, and expanded LIST to allow, let's say, LIST 70,100-200,350 ... etc. Next to be added is 'DELETE' to allow a command such as DELETE 100-200 to delete the named block of lines (with embedded precautions to protect the User).

For the next month or so I shall take time out to concentrate on writing an Instruction Manual for my RBASIC, which should give you all a little time to write to Don about my proposed course in Digital Logic. See you all later.

Sincerely,

Bob

R. Jones
President

Note to Don :

Dear Don,

Over the next few weeks I shall complete my composite of this series on XBASIC, with suitable additions to fill up the disk, at which time I shall submit the whole works to you for the benefit of our reader-friends.

Met Peter Stark and family a few weeks ago. They were out here visiting our EXPO 86 exhibition, and camped overnight in our driveway in their motor-home. We had a long discussion, as did his wife Lois with my wife Joan. A very nice family!

---

microware
MICROWARE SYSTEMS CORPORATION

Contact: Ken Kaplan
(515)224-1929

NEWS RELEASE

NEW TANDY COMPUTER FEATURES OS-9 OPERATING SYSTEM

FOR IMMEDIATE RELEASE

Des Moines, IA — Tandy Corporation has introduced the new Color Computer 3 along with a specialized, high-performance version of Microware's OS-9 Level Two operating system.

The Color Computer 3 combines an economical, high-performance system with an end-user oriented interface. At the nucleus of the machine is an enhanced version of OS-9 Level Two, providing upward compatibility from the Color Computer 2 which uses OS-9 Level One. OS-9 is a real-time, multi-user, multi-tasking operating system that is compact, ROMable and provides a UNIX-style applications environment. The Color Computer version of OS-9 Level Two includes: a multi-screen, multi-window environment that allows several programs to run simultaneously within different windows. It also includes over 30 utility command programs for system and disk file control. OS-9 is available from Tandy for both 128K and 512K versions of the Color Computer 3.

Multi-View is an enhanced windowing environment designed specifically for the Color Computer 3. It gives a common, graphics based, user-friendly environment for application programs to run under. It consists of system support for title bars, menu bars, pull down menus and dialog boxes. The graphics shell allows the user, with the aid of a mouse, to select picture oriented commands to process programs and create windows. Standard desk top utilities include: Calculator, Calendar/Memo Book, Alarm Clock, Printer Configuration, Help, Control Panel and Clipboarding support.

The Color Computer 3's version of Color BASIC, compatible with previous 64K Extended Color BASIC, has been enhanced by Microware for greater control of its new power and capabilities. Commands added to the system allow access to all 512K of memory and hi-resolution graphics. Hi-resolution text and graphics screens are outside the BASIC workspace and do not use up program memory space.

NEW TANDY COMPUTER FEATURES OS-9 OPERATING SYSTEM

A Development Pak, designed to provide advanced utilities for program development, has been simultaneously released by Tandy. Features of the Development Pak include: a relocatable macro assembler and linker, hard disk driver, RAM disk driver, descriptors, system programmer utilities, programming support for utilizing the window environment, plus a screen editor.

OS-9 has long since been considered the defacto standard operating system for 6809 based computer systems, and is now emerging as an important industry standard for Motorola microprocessor based machines. Earlier this year N.V. Philips and Sony announced OS-9 will be the basis for the Compact Disc - Interactive (CD-I) standard. In July, Thomson, Olivetti and Acorn signed an agreement to cooperate in the development of a European Education Standard for 16 bit microcomputers incorporating OS-9. OS-9 has been licensed to over 250 manufacturers for use in a wide variety of industrial, scientific and consumer products.

Founded in 1977, Microware specializes in the development of advanced 68xxx family operating systems and programming languages. Microware offices are located in Des Moines, Iowa and Tokyo, Japan with field representative worldwide.

U.S. Postal Service Statement of Ownership, Management and Circulation (Required by 39 U.S.C. 3685): 1A. Title of Publication: 68 Micro Journal 1B. Publication no: 468510. 2. Date of Filing: 10-01-85: 3. Frequency of Issue: Monthly: 3A. No. of Issues Published Annually: 12. 3B Annual Subscription Price: $24.50 4 and 5: Complete Mailing Address of Known Office, Headquarters or General Business Offices of the Publisher: 3900 Cassandra Smith Rd., Hixson, TN. 37343 6. Full Name and Complete Mailing Address of Publisher: Donald M. Williams Sr., 5900 Cassandra Smith Rd., Hixson, TN 37343. Editor: Donald M. Williams Sr., 5900 Cassandra Smith Rd., Hixson, TN 37343., Managing Editor: Larry E. Williams, 5900 Cassandra Smith Rd., Hixson, TN 37343 7.Owner: Computer Publishing Inc., 3900 Cassandra Smith Rd., Hixson, TN 37343, whose Stockholders are: Donald M. Sr., Frances J. Williams, Larry E.Williams, Mary B.Robertson, Thomas E. Williams. 8. Known Bondholders, Mortgagees, and other Security Holders Owning or Holding 1 Percent or more of Total Amount of Bonds, Mortgages or other Securities: None 9.For Completion By Nonprofit Organizations Authorized To Mail At Special Rates: N/A 10. Extent and Nature of Circulation: A. Total No. Copies (Net Press Run):Average No. Copies Each Issue During Preceding 12 Months: 8900., Actual No. Copies Of Single Issue Published Nearest To Filing Date: 9300. B. Paid and/or Requested Circulation: 1. Sales Through Dealers And Carriers, Street Vendors And Counter Sales: Average No. Copies Each Issue During Preceding 12 Months: 4719. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 5033. 2. Mail Subscriptions: Average No. Copies Each Issue During Preceding 12 Months: 3408. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 3577. C. Total Paid Circulation: Average No. Copies Each Issue During Preceding 12 Months: 8125. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 8610. D. Free Distribution By Mail, Carrier Or Other Means Including Samples, Complimentary, And Other Free Copies: Average No. Copies Each Issue During Preceding 12 Months: 99. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 58. E. Total Distribution (Sum Of C And D): Average No. Copies Each Issue During Preceding 12 Months: 8224, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 8668. F. Copies Not Distributed: 1. Office Use, Left Over, Unaccounted, Spoiled After Printing: Average No. Copies Each Issue During Preceding 12 Months: 462, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 522. 2. Return From News Agents: Average No. Copies Each Issue During Preceding 12 Months: 214, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 110. G. Total:(Sum of E, F1 and 2-Should Equal Net Press Run Shown In A: Average No. Copies Each Issue During Preceding 12 Months: 8900, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 9300. 11. I Certify That The Statements Made By Me Above Are Correct And Complete: (Signed) MARY E. ROBERTSON, Office Manager,68 Micro Journal, Computer Publishing, Inc. 12. For completion by publishers mailing at the regular rates: Permission requested.

Classifieds

As submitted - No Guarantees - As is!

DAISY WHEEL PRINTERS
Qume Sprint 9 - $900
Qume Sprint 5 - $800
TEC FP1500 - $800

Winchester 10 Megabyte Drive - Seagate Model #412 $275.

3 - Dual 8" drive enclosure with power supply. New in box. $125 each.

5 - Siemens 8" Disk Drives. $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS $495.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $250 ready to run.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CU Card - $900 complete.

CDS-1- 20 Meg Hard Disk System with controller $500.

(615) 842-4600 M-F 9 AM to 5 PM EST

***

1 Meg. 2 MHZ SS50 memory board. $400.00 Misc. boards for sale. Call 703-273-6629, Edgar Clark.

***

GIMIX #49, 56K CMOS, 2 para.prt, 2 ser.brd, 8" Qume drives(2), Oki 82A, L/S-ADM36 term., sw/manuals. Make offer: 616-676-7734, Gary Wagner (9-5EDT).

***

SWTPC 69/k 2MHz OS-9 Level One system - two 80tk 5.25" DSDD floppies, PION 512K battery back-up RAM disk, time of day clock, assorted boards, terminal, Microware C and Level One source code package, all original disks and manuals. $875 or any reasonable offer on separate components. (617) 493-5162 Peter.

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative HANDS-ON 68020 computer familiarization two day event. A chance to TRY BEFORE YOU BUY!

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (MUSTANG-020). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00

Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

DATA-COMP

A Division of Computer Publishing, Inc.

5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

Systems available for both OS-9 and UniFLEX. Reservation should be made 15 days in advance. Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attendees should have some working knowledge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Hotel/Motel cost are separate cost, not included in the basic cost shown.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUCCESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program Interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
* All Currency in U.S. Dollars

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

6809<>68XXX UniFLEX

X-TALK

A C-MODEM/Hardware Hookup

Exclusive for the MUSTANG-020 running UniFLEX, is a new transfer program and cable set from DATA-COMP (CPI). X-TALK consist of 2 disks and a special cable, this hook-up enables a 6809 SWTPC UniFLEX computer to port UniFLEX files directly to a 68XXX UniFLEX system.

This is the only currently available method to transfer files, text or otherwise, from a 6809 UniFLEX system to a 68000 UniFLEX system, that we have seen. A must if you want to recompile or cross assemble your old (and valuable) source files to run on a 68000 UniFLEX system. GIMIX users can directly transfer files between a 6809 GIMIX system and our MUSTANG-020 68020 system, or GIMIX 68020 system. All SWTPC users must use some sort of method other than direct disk transfer. The 6809 SWTPC UniFLEX disk format is not readable by most other 68000 type systems.

The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 DB25 connectors. A special SWPTC+ cable and software is also available, at the same price. Orders must specify which type SWTPC 6809 UniFLEX system they intend to transfer from or to.

The X-TALK software is furnished on two disks. One 8" disk containing the 6809 software and one 5" disk containing the 68XXX software. These programs are also complete MODEM programs and can be used as such, including X-on X-off, and all the other features you would expect from a full modem program.

X-TALK can be purchased with/without the special cables, however, this SPECIAL price is available only to registered MUSTANG-020 owners.

X-TALK, w/cable $ 99.95
X-TALK only 69.95
X-TALK w/source $149.95

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

Telephone 615 842-4601
Telex 510 600-6630

Note: Registered MUSTANG-020 owners must furnish system serial number in order to buy at these special low prices.

68' Micro Journal Disks

Disk- 1	Filesort, Miricm, Miricopy,Mirisfms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk- 2	Oldedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3	Chug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6	**Purchase Order, Index (Disk file indx).
Disk- 7	Linking Loader, Rload, Hardware.
Disk- 8	Cross, Lampher (May 82).
Disk- 9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Disassembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initumf68, Testumf68, *Cleanup, *Dskalign, Help, Date.Txt.
Disk-14	*Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven Language Recognition Utility, Anderson March1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Currera.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek.
Disk-30	Simple Winchester, Dec.'86 Green.

NOTE:
This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

8" Disk $14.95 5" Disk $12.95

68' Micro Journal
5900 Cassandra Smith Rd. Hixson, TN 37343

☎ (615)-842-4600

Telex 5106006630

*Indicates 6800
**Indicates BASIC SWTPC or TSC
6809 no Indicator
Foreign Orders Add $4.50 for Surface Mail
or $7.00 for Air Mail
*All Currency in U.S. Dollars

BOARDS & PARTS FOR GIMIX SYSTEMS

CONTACT GIMIX FOR PRICES, DETAILS, AND REQUIRMENTS FOR HARD DISK AND OTHER MASS STORAGE UPGRADES.

THE GIMIX CLASSY CHASSIS #19 consists of a heavyweight aluminum cabinet, constant voltage ferro-resonant power supply, and SS50 Mother board with baud rate generator board $1498.19
#22 Triple Disk Regulator Card $88.22
#93 Baud Rate Generator Board $88.93
#23 Missing Cycle Detector $38.23
#92 Filler Plate ..... $14.92 50 Hz Option $30.00
Cable sets 8" with Back Panel connector $29.25
for two 8" external drives ..... $44.26 for two 5" drives $34.98

CPU BOARDS

#01 GMX III CPU & OS-9 GMX III $1698.01
#02 GMX III CPU & UniFLEX III $1998.02
The #05 GIMIX 6809 PLUS CPU Board $578.05
Options: GIMIX DAT ..... $35.00 9511A $312.00
SWTP DAT ..... $15.00 9512 ..... $265.00
#03 6800 CPU $224.03
#06 6809 CPU w/Timers $288.06
6800 Baud Rate Option ..... Add $30.00

FLOPPY DISK CONTROLLER

#68 DMA $588.68

MEMORY BOARDS FOR 6809/68020 SYSTEMS

#72 256KB CMOS STATIC RAM board with battery back up (specify system) $648.72

MEMORY BOARDS (6800/6809 SYSTEMS ONLY)

#34 8K PROM Card $98.34
#32 16 Socket PROM/ROM/RAM Board, 24 pin $238.32
#31 16 Socket Universal Memory Board, 24/28 pin $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions, freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $498.11
#143 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) $618.13
#15 24K Version of #11, with either large input or output buffers (specify) $648.15

I/O BOARDS (6800/6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.48
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45
#50 I/O for RS-232C, 423, 422-w/6850 ..... $244.50
#52 SSDA with 6852 ..... $254.52
#54 ADLC with 6854 ..... $268.54

CABLES FOR I/O BOARDS — SPECIFY BOARD

#95 Cable sets (1 needed per port) $24.95
#51 Cent. C.P. Cable for #12 & #44 ..... $34.51
#53 Cent. Cable Set $36.53

OTHER BOARDS & PARTS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00
#76 Video Board-80 x 24 ..... $398.78
#08 Relay Driver Package $1126.08
#86 Above without Relays $538.86
Opto Board $348.85
Binder, 3" $12.00
Binder, 2" $9.00

8" DRIVE CABINET & PARTS

2 8" OSDD Drives, Cabinet & Cables 60 Hz only. $1698.88
Cabinet Only for 8" Drive $848.18
220v/50 Hz. Option Add $30.00
Cable Set-Internal for 2 Drives $44.82
Cable Set-Internal for 4 Drives $67.84
Cable from 8" Cab. to Mainframe $45.81
8" Filler Plate $14.83

SOFTWARE:

GIMIX exclusive versions of OS-9/GMX I, II, III & FLEX are for GIMIX hardware only. All versions of OS-9 require the #68 controller. When ordered with controller, FLEX is $30.00
GIMIX versions of FLEX $90.00
GMX VDisk for FLEX 09 ..... $100.00
GMXBUG: PROMs & Manual $148.65
Boot or Video/Boot PROMS (6809) $30.00
GIMIX Boot PROM for UNIFLEX $50.00
RMS (OS9) $250.00
DO (OS9) $70.00
OS-9 GMX III Update w/CPU SPPTROM $125.00
I/O PROMS w/Update $40.00
GMXBUG/FLEX/VDISK w/OS-9 III update ..... Add $175.00
RAM Disk for OS-9 ..... $125.00
O-FLEX $250.00
OS9 GMX I $250.00
OS9 GMX II $500.00
SCULPTOR-6809 (UniFLEX/OS-9) $995.00
SCULPTOR—68020 (UniFLEX) $1595.00

CONTACT GIMIX FOR PRICES AND AVAILABILITY OF OPTIONAL UNIFLEX AND OS9 LANGUAGES AND OTHER SOFTWARE.

ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

LIMITED WARRANTY

GMX INC. ("GMX") Warrants its products against defects in material and workmanship for a period of ninety days from the date of shipment. The obligation of GMX is limited to the repair or replacement of any product, free of all charges, which proves defective during this period. This warranty does not cover damage due to accidents, negligence, abuse or tampering.

GMX MAKES NO OTHER WARRANTIES OR GUARANTEES, EXPRESS, STATUTORY, OR IMPLIED, OF ANY KIND WHATSOEVER WITH RESPECT TO ANY PRODUCT PURCHASED, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS HEREBY DISCLAIMED BY GMX AND EXCLUDED FROM ANY AGREEMENT MADE BY GMX.

GMX will not be responsible for any damage of any kind

not covered by the exclusive remedies set forth in this limited warranty. GMX will not be responsible for any special, indirect, or consequential damage caused by its products.

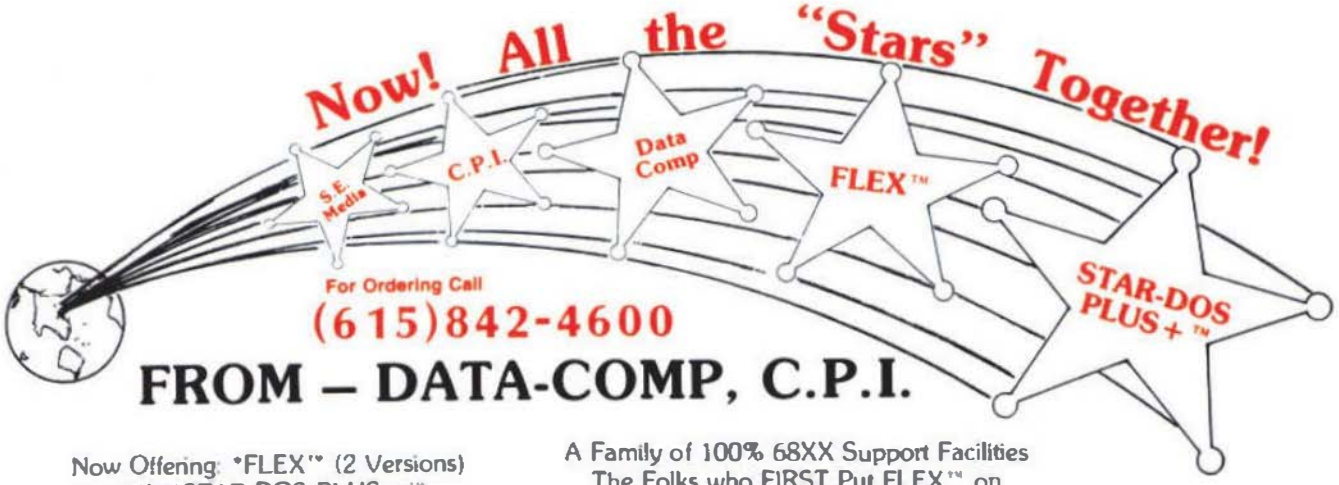
GMX products are not for consumer use. GMX expressly disclaims all warranties on any of its products which may be included in any product normally used for personal or family purposes.

Contact GMX by mail at 1337 West 37th Place, Chicago, IL 60609; or phone at (312) 927-5510; if your product is defective to arrange for its repair or replacement under this warranty.

Repair charges for GIMIX products after warranty period will be $35.00 per hour per board (minimum $35.00) plus parts. Customer pays freight charges both ways. If GIMIX determines that replacement is desirable instead, we will notify you. Charges for checking out complete system will be $500.00 plus parts, freight, and necessary board repairs.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX inc. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

S - 50 BUS / 68XX

Board and/or Computer
Terminals-CRTs-Printers
Disk Drives-etc.

REPAIRS

This

Not This

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to **'ALL'** S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. *Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and *confirm a Data-Comp service & shipping number*! This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but **NO** advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates *must be requested*. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with the item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4607
Telex 5106006630